WebSphere® software

IBM

Linux
Application Development Using WebSphere Studio 5

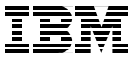
- A comprehensive guide to Linux support of WebSphere development
- Develop, test, and deploy your Web application on Linux
- Setting up your Linux enviroment



Osamu Takagiwa
Bruno Hulbusch
Markus Mueller
Chetan Pattni
Wolfgang Sanyer

Redbooks

ibm.com/redbooks



International Technical Support Organization

Linux Application Development Using WebSphere Studio 5

March 2003

SG24-6431-00

Note: Before using this information and the product it supports, read the information in “Notices” on page xiii.

First Edition (March 2003)

This edition applies to WebSphere Studio Application Developer for Linux Version 5 and WebSphere Application Server for Linux Version 5.

© Copyright International Business Machines Corporation 2003. All rights reserved.
Note to U.S. Government Users Restricted Rights -- Use, duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corp.

Contents

Figures vii

Tables xi

Notices xiii
Trademarks xiv

Preface xv
The team that wrote this redbook xv
Become a published author xvii
Comments welcome xvii

Chapter 1. What is Linux? 1
1.1 Linux as an operating system 2
1.1.1 Linux is reliable 2
1.1.2 Linux is cheaper 2
1.1.3 Linux is portable 2
1.1.4 Linux is easy-to-use 3
1.1.5 Linux is powerful 3
1.2 IBM and Linux 3
1.3 Web development on Linux 4

Chapter 2. WebSphere application development for Linux 5
2.1 The IBM Framework for e-business 6
2.2 Models of the Framework 7
2.2.1 The system model 7
2.2.2 The programming model 8
2.3 WebSphere Application Developer and Server 9
2.3.1 WebSphere products 11
2.3.2 WebSphere Studio Application Developer for Linux 12
2.3.3 WebSphere Application Server for Linux 13

Chapter 3. Setting up the development environment 15
3.1 Workbench fundamentals 16
3.1.1 Resource perspective 16
3.1.2 Java perspective 18
3.1.3 Web perspective 19
3.1.4 J2EE perspective 21
3.1.5 Server perspective 21

© Copyright IBM Corp. 2003. All rights reserved.

3.1.6 XML perspective . . . 23
3.1.7 Data perspective . . . 23
3.1.8 Debug perspective . . . 24
3.1.9 Profiling perspective . . . 25
3.1.10 Team perspective . . . 26
3.1.11 Help perspective . . . 27
3.1.12 Workbench views . . . 28
3.1.13 Workbench projects . . . 28
3.2 Sample application . . . 29
3.2.1 Web application using HTML, JSP, servlets, and JavaBeans . . . 30
3.2.2 Using Enterprise JavaBeans with your Web application . . . 31
3.2.3 Generating Web application using XML . . . 32
3.2.4 Testing and deploying Web application . . . 32
3.2.5 Database design for ITSO Bank application . . . 33

Chapter 4. HTML, JSP, servlet, JavaBeans, and database . . . 35
4.1 Preparing for development . . . 36
4.1.1 Creating a new project . . . 37
4.2 HTML . . . 41
4.3 JSP . . . 43
4.4 Servlet . . . 46
4.5 JavaBeans . . . 50
4.6 Database . . . 53
4.6.1 ITSO Bank database . . . 53
4.6.2 Connecting to a database from Application Developer . . . 55
4.6.3 Using SQL Query Builder in Application Developer . . . 56

Chapter 5. Enterprise JavaBeans 2.0 . . . 59
5.1 The types of Enterprise JavaBeans . . . 60
5.1.1 Java Message-driven Beans . . . 60
5.1.2 EJB 2.0 Bean Managed Persistence Entity Bean . . . 78
5.2 ITSO Bank bean sample . . . 98

Chapter 6. The eXtensive Markup Language . . . 113
6.1 XML tools in WebSphere Application Developer . . . 114
6.2 Introducing ITSO Banking example using XML . . . 115
6.3 Using the wizards to create XML from SQL . . . 117
6.3.1 RDB to XML mapping . . . 117
6.3.2 Create a SQL query . . . 118
6.3.3 Generate XML from SQL query . . . 119
6.3.4 XML, DTD, and XSL editors . . . 121
6.4 Dynamically generating XML from SQL . . . 124
6.4.1 Setting up a Web project . . . 125
6.4.2 Walking through the Web application . . . 126

6.5 Using the XSL debugger and transformation tools 129
6.6 Motivation to use XML/XSL instead of JSP 132
6.6.1 Struts with JavaServer Page drawbacks 132

Chapter 7. Building a Web application with Ant 135
7.1 Philosophy of Ant 136
7.1.1 Build process approaches 136
7.2 Setting up your environment to use Ant 137
7.2.1 Basics of using Ant 138
7.3 Building J2EE applications with Ant 140

Chapter 8. Deploying the Web application 151
8.1 Deploying an Enterprise Application manually 152
8.1.1 EAR export from WebSphere Studio Application Developer 152
8.1.2 Starting the WebSphere administration console 153
8.1.3 Configuration WebSphere resources 155
8.1.4 Installation of the ITSO Bank EAR file 159
8.1.5 Testing the application 162
8.2 Setting up a remote server 162
8.2.1 IBM Agent Controller 163
8.2.2 Creating a server for remote testing with Application Server 163
8.3 Automatic deployment by tools 170
8.3.1 Installing application with the wsadmin tool 170
8.3.2 Control the Application Server 172
8.3.3 Deployment with Ant 172

Appendix A. Installation instructions 177
How to install Linux 178
How to install WebSphere Application Developer 179
How to install WebSphere Application Server 182
How to install IBM DB2 183
How to configure CVS 186
How to configure Telnet, FTP, and Samba 187

Appendix B. Additional material 189
Locating the Web material 189
Using the Web material 189
System requirements for downloading the Web material 190
How to use the Web material 190

Abbreviations and acronyms 191

Related publications 193
IBM Redbooks 193

Other resources 193
Referenced Web sites 194
How to get IBM Redbooks 194
IBM Redbooks collections 194

Index 195

Figures

0-1 Markus, Bruno, Chetan, Wolfgang, and Osamu xvi
2-1 3-tier system model 7
2-2 IBM e-business Framework 8
2-3 WebSphere platform pyramid 9
3-1 Select perspective 17
3-2 Resource perspective 18
3-3 Java perspective 19
3-4 Web perspective 20
3-5 J2EE perspective 21
3-6 Server perspective 22
3-7 XML perspective 23
3-8 Data perspective 24
3-9 Debug perspective 25
3-10 Profiling perspective 26
3-11 Team perspective 27
3-12 ITSOBankSelectWeb project model 30
3-13 ITSOBankSelectUpdate project model 30
3-14 ITSOBankEJB project model 31
3-15 ITSOBankXML project model 32
3-16 Deployment using Ant 33
3-17 ITSOBankDatabase project model 34
4-1 ITSOBankSelectWeb model 36
4-2 ITSOBankUpdateWeb model 37
4-3 ITSOBankSelectEar directory 38
4-4 ITSOBankSelectWeb directory 39
4-5 ITSOBankWeb login page 43
4-6 ITSOBankWeb account information page 46
4-7 ITSOBankWeb account update page 46
4-8 ITSOBankWeb confirmUpdate page 53
4-9 ITSO Bank example database 55
4-10 ITSOBankWebConnection directory 56
4-11 SQL joins window 57
5-1 JMS class hierarchy 61
5-2 Enable the administration client 63
5-3 Add a new listener port 64
5-4 WebSphere JMS Provider options 66
5-5 Add WASQueueConnectionFactory dialog 67
5-6 Add WASQueue dialog 68

© Copyright IBM Corp. 2003. All rights reserved.

5-7 Select 2.0 EJB types 70
5-8 Define EJB details for a Message-driven Bean 71
5-9 A simple asynchronous messaging scenario 77
5-10 The ITSO Bank sample configuration 99
6-1 Flow of the RDB to XML mapping 117
6-2 Input of specify variables 118
6-3 Result of the SQL query in the SQL builder 119
6-4 XML From An SQL Query Wizard 120
6-5 Generated files for customerBalance example 121
6-6 customerBalance.html 123
6-7 Servlet access to get XML document 124
6-8 Setting up the environment 125
6-9 Login form page 128
6-10 Result page account balances (default) 128
6-11 Login form page 129
6-12 Result page of account balances (ITSO Banking example style) 129
6-13 XSL and XML file for the transformation into HTML 130
6-14 XSL Debug perspective 131
7-1 Flie system structure of the build directory 141
8-1 Exporting the EAR file 153
8-2 First Steps Tool Tips 154
8-3 Administration Console login 155
8-4 WebSphere Administration Console Version 5 155
8-5 JDBC Provider 156
8-6 Additional Properties: data sources 157
8-7 Custom Properties 157
8-8 Change the database name value 157
8-9 J2C Authentication Data Entries 158
8-10 Create a new authentication alias calls itsops 158
8-11 Set the authentication aliases in the Data Source frame 158
8-12 Environment: WebSphere variables 159
8-13 DB2 variable 159
8-14 Install new application 160
8-15 Preparing for application install 160
8-16 Step 1: Deployment options 161
8-17 Step 3: Provide JNDI names for beans 161
8-18 Creating a remote server instance 164
8-19 Setting the remote server host address 165
8-20 Remote server instance settings 166
8-21 Remote file transfer option 167
8-22 Remote copy options 168
8-23 FTP configuration options 169
8-24 Ant invokes the Application Server tools 173

A-1 DB2 Setup started window 183
A-2 DB2 Installing window 185
A-3 DB2 Setup finished window 186

Tables

4-1 Customer table 53
4-2 Address table 54
4-3 Customer account table 54
4-4 Account table 54
4-5 Check_Reorder table 54
5-1 Data access modules for DB2 89
7-1 Build-In tasks of ITSO Banking example 139
7-2 Mapping J2EE project to CVS module 140
7-3 Target names of build.xml 141

© Copyright IBM Corp. 2003. All rights reserved.

Notices

This information was developed for products and services offered in the U.S.A.

IBM may not offer the products, services, or features discussed in this document in other countries. Consult your local IBM representative for information on the products and services currently available in your area. Any reference to an IBM product, program, or service is not intended to state or imply that only that IBM product, program, or service may be used. Any functionally equivalent product, program, or service that does not infringe any IBM intellectual property right may be used instead. However, it is the user's responsibility to evaluate and verify the operation of any non-IBM product, program, or service.

IBM may have patents or pending patent applications covering subject matter described in this document. The furnishing of this document does not give you any license to these patents. You can send license inquiries, in writing, to:
IBM Director of Licensing, IBM Corporation, North Castle Drive Armonk, NY 10504-1785 U.S.A.

The following paragraph does not apply to the United Kingdom or any other country where such provisions are inconsistent with local law: INTERNATIONAL BUSINESS MACHINES CORPORATION PROVIDES THIS PUBLICATION "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF NON-INFRINGEMENT, MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. Some states do not allow disclaimer of express or implied warranties in certain transactions, therefore, this statement may not apply to you.

This information could include technical inaccuracies or typographical errors. Changes are periodically made to the information herein; these changes will be incorporated in new editions of the publication. IBM may make improvements and/or changes in the product(s) and/or the program(s) described in this publication at any time without notice.

Any references in this information to non-IBM Web sites are provided for convenience only and do not in any manner serve as an endorsement of those Web sites. The materials at those Web sites are not part of the materials for this IBM product and use of those Web sites is at your own risk.

IBM may use or distribute any of the information you supply in any way it believes appropriate without incurring any obligation to you.

Information concerning non-IBM products was obtained from the suppliers of those products, their published announcements or other publicly available sources. IBM has not tested those products and cannot confirm the accuracy of performance, compatibility or any other claims related to non-IBM products. Questions on the capabilities of non-IBM products should be addressed to the suppliers of those products.

This information contains examples of data and reports used in daily business operations. To illustrate them as completely as possible, the examples include the names of individuals, companies, brands, and products. All of these names are fictitious and any similarity to the names and addresses used by an actual business enterprise is entirely coincidental.

COPYRIGHT LICENSE:
This information contains sample application programs in source language, which illustrates programming techniques on various operating platforms. You may copy, modify, and distribute these sample programs in any form without payment to IBM, for the purposes of developing, using, marketing or distributing application programs conforming to the application programming interface for the operating platform for which the sample programs are written. These examples have not been thoroughly tested under all conditions. IBM, therefore, cannot guarantee or imply reliability, serviceability, or function of these programs. You may copy, modify, and distribute these sample programs in any form without payment to IBM for the purposes of developing, using, marketing, or distributing application programs conforming to IBM's application programming interfaces.

© Copyright IBM Corp. 2003. All rights reserved.

Trademarks

The following terms are trademarks of the International Business Machines Corporation in the United States, other countries, or both:

AIX®
Balance®
CICS®
Database 2™
DB2®
Everyplace™
IBM®
IMS™
MQSeries®
MVS™
Perform™
Redbooks™
SP™
SP1®
Tivoli®
VisualAge®
WebConnection®
WebSphere®
zSeries™
Approach®
Domino™AIX®
CICS®
Database 2™
DB2®
Everyplace™
IBM eServer™
IBM®
IMS™
MQSeries®
MVS™
Redbooks (logo)™ 
Redbooks™
Tivoli®
VisualAge®
WebConnection®
WebSphere®
zSeries™

The following terms are trademarks of other companies:

ActionMedia, LANDesk, MMX, Pentium and ProShare are trademarks of Intel Corporation in the United States, other countries, or both.

Microsoft, Windows, Windows NT, and the Windows logo are trademarks of Microsoft Corporation in the United States, other countries, or both.

Java and all Java-based trademarks and logos are trademarks or registered trademarks of Sun Microsystems, Inc. in the United States, other countries, or both.

C-bus is a trademark of Corollary, Inc. in the United States, other countries, or both.

UNIX is a registered trademark of The Open Group in the United States and other countries.

SET, SET Secure Electronic Transaction, and the SET Logo are trademarks owned by SET Secure Electronic Transaction LLC.

Other company, product, and service names may be trademarks or service marks of others.

Preface

Linux is the fastest-growing server operating system in the world because of its powerful functionality, rock-solid stability, and open source foundation. Applications developed on Linux are reliable, portable, and cost efficient. This IBM Redbook helps you get familiar with IBM middleware and tools for Linux, and develop your new Web application on Linux. The book is aimed to show IBM's ability to provide an advanced platform for WebSphere application development using Linux as the operating system.

The approach we have taken is to build an ITSO Banking example that has a backend database and a frontend e-business banking application. The Linux distribution that we use is Red Hat Linux Version 7.3.

This book also shows you how to install the software to set up your development environment.

The team that wrote this redbook

This redbook was produced by a team of specialists from around the world working at the International Technical Support Organization, Raleigh Center.

Osamu Takagiwa is an Advisory IT Specialist at the International Technical Support Organization, San Jose Center. He writes extensively and teaches IBM classes worldwide on all areas of application development. Before joining the ITSO two years ago, Osamu worked at IBM Japan as an I/T Specialist.

Bruno Hulbusch is a European DBA working in the Application Services Group for Ford Motor Company in Germany. He has more than 14 years experience in system programming for networking components in VSE, VM, and MVS. He has 5 years of experience in writing Web applications for marketing and sales. During the last two years, he has concentrated on new technologies on the mainframe, such as Web services, Linux, and J2EE application serving. Bruno holds a bachelor's degree in Electronic Engineering.

Markus Mueller is a Senior IT Specialist with the IBM's subsidiary, SerCon GmbH in Frankfurt, Germany. He has been with IBM for five years. His areas of expertise include AIX, Java, e-business, and e-infrastructure solutions. He specializes in software management, development, and deployment. Markus holds a bachelor's degree in Electrical Engineering.

© Copyright IBM Corp. 2003. All rights reserved.

Chetan Pattni is an IT Specialist with IBM Global Services in Toronto, Canada. He has been with IBM for three years. His areas of expertise include Web application design and development. He specializes in installing and configuring network solutions. Chetan holds a bachelor's degree in Computer Science from McMaster University.

Wolfgang Sanyer is a certified Consulting IT Specialist at IBM North America. He has over twelve years of experience in object-oriented technology in all phases of software development. Wolfgang is currently part of the elite group WebSpeed (East) providing his expertise in a sales environment to high profile customers. Wolfgang's previous experiences involve using VisualAge UML Designer; a published redbook in1998; a speaker at the WebSphere Conference 2002; teaching at Wake Technical College, Raleigh. Wolfgang holds a Bachelor of Science degree in Computer Science from Polytechnic University in New York. His areas of expertise include client/server architecture development, artificial intelligence, object-oriented development, and eXtreme Programming.



Figure 0-1 Markus, Bruno, Chetan, Wolfgang, and Osamu

Thanks to the following people for their contributions to this project:

Mark Endrei
International Technical Support Organization, Raleigh Center

Become a published author

Join us for a two- to six-week residency program! Help write an IBM Redbook dealing with specific products or solutions, while getting hands-on experience with leading-edge technologies. You'll team with IBM technical professionals, Business Partners and/or customers.

Your efforts will help increase product acceptance and customer satisfaction. As a bonus, you'll develop a network of contacts in IBM development labs, and increase your productivity and marketability.

Find out more about the residency program, browse the residency index, and apply online at:

ibm.com/redbooks/residencies.html

Comments welcome

Your comments are important to us!

We want our Redbooks to be as helpful as possible. Send us your comments about this or other Redbooks in one of the following ways:

- Use the online **Contact us** review redbook form found at:

 ibm.com/redbooks

- Send your comments in an Internet note to:

 redbook@us.ibm.com

- Mail your comments to:

 IBM Corporation, International Technical Support Organization
 Dept. HZ8 Building 662
 P.O. Box 12195
 Research Triangle Park, NC 27709-2195

1

What is Linux?

In this chapter, we discuss the following sections:

- Linux as an operating system: Provides background information on Linux, its origins, performance, usability, and advantages of using Linux.
- IBM and Linux: Shows IBM's commitment to Linux as one of its supported operating systems.
- Web development on Linux: Provides background information on this redbook and the project.

© Copyright IBM Corp. 2003. All rights reserved.

1.1 Linux as an operating system

UNIX is an ancestor of Linux. It is an operating system that was created by *Linus Trovals*, a university student in Finland in 1991. He wrote the kernel based on the minix to make it like a UNIX-like operating system for desktops.

Operating systems are simply software applications that control the computer. Most people use Microsoft Windows as their operating system. However, unlike the Windows operating system, Linux is much more reliable, cheaper, portable, easy-to-use, and powerful.

1.1.1 Linux is reliable

Since Linux is written by many different programmers, it is known to have fewer reported bugs, and it is much more stable than any other operating system. Application failure on most operating systems requires a reboot of the system, but for Linux the only time you would turn off the Linux system is to add new hardware, or to boot from a different kernel. Also, Linux servers are famous for their ability to run for hundreds of days, compared to the regular reboots required with a Windows system. This tells the end-user that Linux is very reliable.

1.1.2 Linux is cheaper

Open source software, compared to the traditional commercial software, has a huge impact on the computer industry. Once you purchase Microsoft Windows it has guidelines that must be followed for use within a business or at home. Also, it is impossible to get an upgrade without spending enormous amounts of time and money. Linux on the other hand, can be found at no charge from the Internet. Once you have become a part of the Free Software Foundation (FSF) you can copy GNU software and give it to anyone you choose. Furthermore, you can modify and distribute an improved version of the GNU software without any restrictions. The open source promotes security by allowing anyone to review the code and modify it as required. As the fixes are generated, they can be distributed much faster than commercial software.

1.1.3 Linux is portable

Many users like to have platform, vendor, and application independence. For example, the Microsoft Windows operating system cannot be loaded on just any computer. It has to meet very specific installation requirements, and the platform must be a variation of the Intel processor. Linux on the other hand, has a minimum specification, however, it is platform independent. It can be installed on a DEC Alpha, SUN Sparc, or 386/486-based computers. Just about any computer can easily adapt to Linux.

1.1.4 Linux is easy-to-use

UNIX Command Line Interface (CLI) was not appealing to many users, so Massachusetts Institute of Technology (MIT) released X to the public in 1988. MIT passed the development of X to the X Consortium. Within the X Consortium, the Xfree86 Project, Inc. now freely distributes the implementation of the X-Windows system. X-Windows is a graphical user interface that can run on Linux. X has two main advantages: it lets you configure the desktop to the *n*-th degree, and it allows users to access local and remote applications while using very little resources.

1.1.5 Linux is powerful

Since Linux is capable and effective as an operating system, many users have come to think of Linux as being powerful. The cost of Linux compared to other operating systems is minimal. The stability has proven to be acceptable by many users in the computing environment. The constant headaches that are caused when an operating system crashes is clearly not the case when using Linux. Furthermore, to be able to move from one platform to another without many difficulties is a major advantage in today's changing environment. Also, Linux can work on just about any computer system. GNOME and K Desktop Environment (KDE) are free and easy-to-use desktop environments for all users, and they are also powerful application frameworks for many developers. Therefore, the combination of the cost, stability, portability, and ease-of-use makes Linux powerful.

1.2 IBM and Linux

As stated in the IBM Web site, http://www.ibm.com/linux Linux is an open source platform that is stable, secure, and powerful. Some of the major players in the development of Linux are Red Hat, SuSE, Turbolinux, and Caldera Systems. IBM joined in to lay down the groundwork for co-marketing, development, training, and support initiatives, which encourage customers to use Linux for their business. IBM is committed to offering full hardware and software support for Linux to all IBM customers and business partners. Linux users can use IBM's experience, resources, and skills to satisfy their technical needs. The open platform environment for running e-business applications enables IT professionals to build Web solutions independent of specific vendor software and hardware.

Although there are many IBM products that are available for Linux, we focus on application development using WebSphere Version 5.0 on Red Hat Linux, Version 7.3.

1.3 Web development on Linux

This book is produced from a residency at the International Technical Support Organization (ITSO) in San Jose, California. The book is aimed to show IBM's ability to provide an advanced platform for WebSphere Application Development using Linux as the operating system.

The approach we have taken is to build an ITSO Banking example that has a backend database and a frontend e-business banking application. The Linux distribution that we use is Red Hat Linux, Version 7.3. The IBM software we use is:

- WebSphere Studio Application Developer, Version 5.0
- WebSphere Application Server, Version 5.0
- IBM HTTP Server, Version 1.3.19.3
- IBM Database 2, Version 7.2, FixPak 7
- IBMJava2, Version 1.3.1
- IBM Remote Agent

The ITSO Banking example is used to explain Web development using WebSphere on Linux. The following topics are covered in this book:

- Dynamic Web page using servlet, JavaServer Pages (JSP), and JavaBeans
- Enterprise JavaBeans with Message Driven Beans, and Session and Entity Beans
- Development using eXtensible Markup/Stylesheet Language (XML/XSL) with JSP
- Building and deploying using Ant

In addition to the above, this book also shows you how to install the software to set up your development environment.

2

WebSphere application development for Linux

In this chapter, we discuss the following sections:

- The IBM Framework for e-business provides background information for e-business and its underlying principles.
- Models of Framework explain the basic principles of the system and programming models.
- WebSphere Application Developer and Server provide background information on WebSphere, its products, and what is new with WebSphere Application Developer and Server for Linux.

© Copyright IBM Corp. 2003. All rights reserved.

2.1 The IBM Framework for e-business

Here we give an overview and explain what the Framework can do for you.

When building a successful e-business application, you require an e-business Framework. A Framework environment consists of guidelines to help develop e-business applications. The high-level architecture element of the Framework includes the system and programming models. IBM has already established a Framework for e-business, and has made it part of the core of IBM's e-business software strategy. The Framework consists of products and solutions to help customers build, run, and manage successful e-business applications.

IBM's Framework for e-business consists of the following:

- A commitment to accept and further industry standards, while providing a multi-platform and multi-vendor solution.
- Offer a design, development, and deployment model that guides you through the process of application development.
- State-of-the-art products that allow you to build and run e-business applications.

As customers adopt to the Framework, they have access to many products and solutions. Here are some of the advantages of using the IBM Framework for e-business:

- Reduce development time of an e-business application by using existing solutions and state-of-the-art products.
- Support for heterogeneous client environments and integration standards, such as XML and Component Object Request Broker Architecture (CORBA)/Internet Inter-ORB Protocol (IIOP) which allows exchange of data between applications and systems.
- Support application protocols such as Simple Mail Transfer Protocol (SMTP) and Post Office Protocol (POP)/ Internet Message Access Protocol (IMAP) which are used for e-mail and management applications.
- Support communication standards such as Secure Socket Layer (SSL) and Transmission Control Protocol, or Internet Protocol (TCP/IP).

 And last but not least:

- IBM's commitment to support and enhance open industry standards.

The IBM Framework for e-business helps you to deploy and develop new or existing business applications, so that you can quickly operate a dynamic e-business.

2.2 Models of the Framework

As mentioned earlier, The high-level architecture element of the Framework includes the system and programming models. We will describe these models since they are the core of any development environment.

2.2.1 The system model

The Framework is based on an *n*-tier environment where a number of tiers of application logic and business services are separated into components that communicate with each other across a network. The basic form is a logical three-tier computing model.

Figure 2-1 show a high-level system model for running an e-business application.

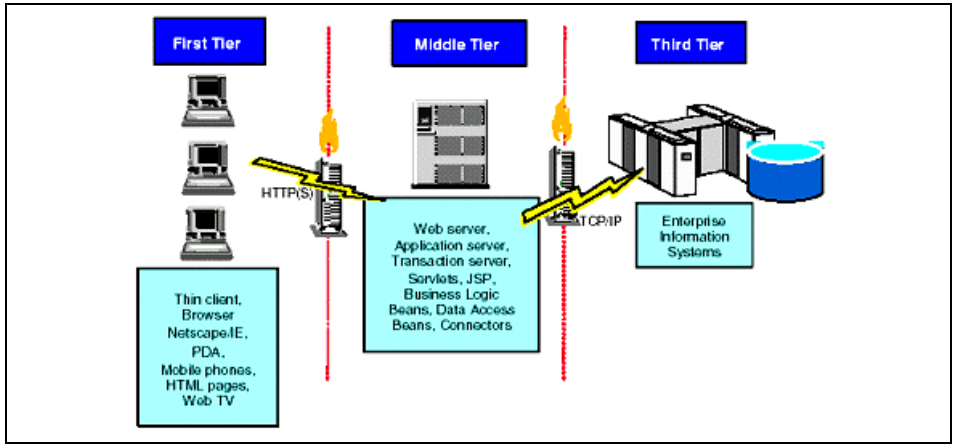


Figure 2-1 3-tier system model

We will briefly describe the three tiers:

- **The first tier**: This consists of the clients. Their main function is to present information and results produced by an application to the user. In this system model, the clients are sometimes referred to as *thin clients*. This means that little or no application logic is executed on the client, hence, relatively little software is required to be installed on the client. The common element that ties these clients to the Web application server is their implementation of a set of widely supported, Internet-based technologies and protocols; along with Java, which enables them to provide interaction between users and applications.
- **The second tier**: This tier has a standards-based Web server that interacts with the client tier and defines user interaction. The Web application server

executes business logic independently of the client type and user interface style. It is implemented using various Internet and Java technologies, including the Hypertext Transfer Protocol (HTTP) server, and the Enterprise Java services, which enable rapid development and deployment of applications in a distributed network environment. Java servlets, JavaServer Pages, and Enterprise JavaBeans are examples of the components deployed in the Web application server. These server-side components communicate with their clients and other application components via HTTP or IIOP, and use the directory and security services provided by the network infrastructure. They can also leverage database, transaction, and groupware facilities.

- **The third tier**: This tier mainly consists of Customer Information Control System (CICS) server, legacy applications developed on mainframes, legacy systems, or relational databases such as DB2.

2.2.2 The programming model

The application programming model highlights the software tools and products used to build, run, and manage e-business applications.The Framework is constantly evolving to support more industry standards, expand the product offerings, and make the methodologies stronger.

Figure 2-2 shows the Framework for e-business.

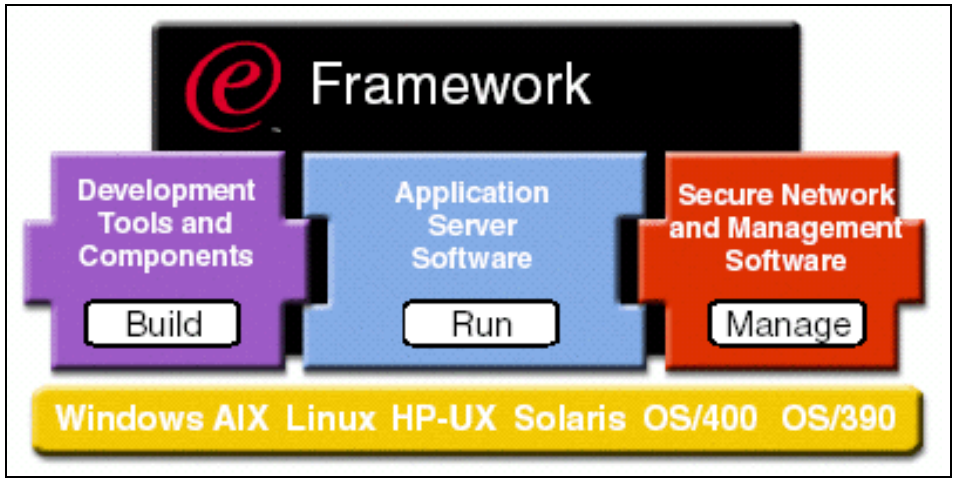


Figure 2-2 IBM e-business Framework

We will briefly describe the three parts of the Framework:

- Development Tools and Components: This component consists of the build tools. For example, WebSphere Studio or Domino Designer.
- Application Server Software: The application servers include runtime environments that are available on all major platforms. The application server software should include collaboration, transaction, database, and integration.
- Secure Network and Management Software: The Framework includes a complete portfolio of security and management products and services such as Tivoli.

We have given you an introduction to the building blocks that are required when developing an e-business application. In the next section, we will elaborate on the WebSphere software platform, and the new additions to the WebSphere family.

2.3 WebSphere Application Developer and Server

WebSphere is an infrastructure for building e-business applications. WebSphere has evolved to meet the demands of the changing business environment by increasing the developer's ability to build very complex solutions, and allowing tool integration to an open platform. Based on an Eclipse Framework, WebSphere has come a long way to provide everything a customer needs to build, deploy, and operate a dynamic e-business.

Figure 2-3 is the WebSphere platform pyramid.

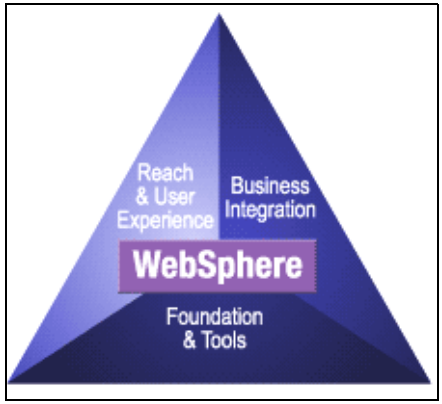


Figure 2-3 WebSphere platform pyramid

The WebSphere pyramid consists of the *Foundation and Tools*, *Reach and User Experience*, *Business Integration,* and *Transaction Servers and Tools*. Together these components close the gap between business strategy and information technology. We will discuss each component below:

Foundation and tools

If you want to make a presence on the Internet, you will require a solid foundation and the latest tools. Having business processes and transactions on the Web allows your customers, partners, and employees to buy, sell, and share information anytime, anywhere. The WebSphere e-business infrastructure is about Web-enabling your business as fast as possible. For example, as stated in the IBM WebSphere Web site:

> "...the WebSphere Application Server, Version 5 provides rich e-business application deployment capabilities for transaction management in a heterogeneous environment, comprehensive Web services support, increased security..."

Reach and user experience

This can happen only after your e-business starts to grow. WebSphere e-business infrastructure allows you to provide any user or device access to customized content, therefore, allowing you to extend and personalize your e-business.

Business integration

This is all about acquiring business agility by integrating and automating business processes. Delivering a customer's e-business needs requires your business to have both the internal and external business processes integrated in the e-business environment. Once the employees, customers, suppliers, business partners, and e-marketplaces can connect with speed, efficiency, and agility, they can now provide dynamic e-business solutions.

Transaction servers and tools

WebSphere infrastructure has the necessary features to integrate traditional assets, so that companies are not left wondering what they will do as the new technology revolutionizes the way people work and interact. As stated in the IBM WebSphere Web site:

> "Leveraging these core software assets allows you to capitalize on existing investments, shorten deployment time for new e-business applications, and create real competitive advantage."

We have discussed briefly the WebSphere software platform. For more detailed information visit the IBM WebSphere Web site:
http://www.ibm.com/software/websphere

2.3.1 WebSphere products

As shown above, the WebSphere platform is divided in four main components. Each of these components consists of products that are used for development.

- The *Reach and User Experience* component consists of the following products:
 - IBM WebSphere Commerce helps you offer products and services over the Internet, which promotes a global and mobile marketplace.
 - IBM WebSphere Portal is a single point of interaction for dynamic information, applications and processes.
 - IBM Pervasive Products has products such as WebSphere Voice Server or DB2 Everyplace, which allows you to deliver any information over various network, using different devices.
- The *Transaction Servers and Tools* component consists of the following products:
 - IBM Enterprise Modernization solutions, which use the IBM WebSphere software to help you develop the process, tooling, and infrastructure to transform your business into the world of e-business.
 - Transaction Processing Product has products such as IBM's CICS Transaction Server, and other similar products that can handle more than thirty billion transactions per day.
 - Traditional products contain enterprise application development and operational tools, which improve user productivity and effectiveness.
- The *Foundation and Tools* component consists of the following products:
 - IBM WebSphere Application Server is a world-class infrastructure for open e-business applications, which provides a complete set of e-business application deployment and integration services.
 - IBM WebSphere Studio is an integrated application development environment, which makes building the e-business application easier and faster.
 - IBM WebSphere Host Integration is a single offering, which provides a fast and cost-effective way to access to publish host information to Web-based clients and applications.
- The *Business Integration* component consists of the following products:

- Process integration products such as IBM WebSphere MQ Workflow, which provides ability to design, document, execute, or optimize the business processes.
- Application connectivity has products such as IBM WebSphere MQ, which is a key product for dynamic integration. They provide flexible and reliable connections between applications.

We urge you to visit the IBM WebSphere Web site for more detailed information, and make use of these tools in your next e-business application:
http://www.ibm.com/software/websphere

2.3.2 WebSphere Studio Application Developer for Linux

IBM WebSphere Studio Application Developer for Linux, also known as WebSphere Application Developer, is an integrated application development environment. For IBM this “represents the next generation of IBM offerings that make building e-business application easier and faster.”

What is new in Version 5

Some of the new features of this version are:

- Support for J2EE 1.3, EJB 2.0, Servlet 2.3 and JSP 1.2
- Support for both WebSphere Application Server Version 4 (J2EE 1.2) and Version 5 (J2EE 1.3)
- Allow users for WebSphere Application Server Version 4 to adopt to Eclipse Version 2 without any upgrades
- Uses the Eclipse Version 2 platform

Version 5 feature

We will briefly describe some of the version features:

- J2EE development environment: Some of the features include the development environment, which includes support for J2EE 1.3 specifications with EJB 2.0, servlet 2.3 and JSP1.2. Ability to generate code on both J2EE 1.2 and 1.3 specifications. EJB-to-RDB mapping tools for editing the mapping between EJBs and relational database tables with top-down, bottom-up, and meet-in-the-middle capability. Tools for creating, editing, and validating EAR files.
- Web services development environment: Some of the features include support for UDDI Version 2; browse the UDDI business registry to locate existing Web services for integration; support for Web Services Inspection Language (WSIL); and testing Web services as they run locally or remotely.

- XML development environment: Some of the features include debug and edit XSL with code assistance. Define mappings between relational table and DTD files. Create and execute XPath using XPath Wizard.
- Web development environment: Some of the features include support for XHTML. JavaScript editing and validating. Cascading Style Sheet (CSS) editing support. HTTP/FTP import and Web project creation using the J2EE container structure.
- Team collaboration: Some of the features include managing project versions; serving as a backup vehicle. New support for namespace versions so that when files are moved, deleted, or renamed, Relation ClearCase LT will perform the equivalent SCM operation.

2.3.3 WebSphere Application Server for Linux

As part of the foundation for the WebSphere software platform, the WebSphere Application Server for Linux maintains its reputation as the premier Java-based application platform for the dynamic e-business environment. It provides a very rich e-business application deployment environment for transaction management, Web services, security, performance, availability, connectivity, and scalability.

What is new with Version 5

Some of the new features of this version are:

- A fast, scalable, and reliable server based on the Java 2 Platform, Enterprise Edition (J2EE) technology
- Compatible to J2EE 1.3 technology
- Supports core Web services standards like XML, Simple Object Access Protocol (SOAP), and Web Services Description Language (WSDL)
- Offers deployment of Web services using SOAP and HTTP, JMS or Remote Method Invocation Internet Inter-ORB Protocol (RMI/IIOP) communication mechanism
- Delivers dynamic caching, content-based routing, load balancing, and content distribution for application optimization

2.3.4.2 Version 5 features

We will briefly describe some of the version features:

- Enable dynamic application integration - Dynamic application integration through native, high-performance Java Messaging Services (JMS), J2EE 1.3 Message Beans and container managed messaging.

- Web services - Allows you to create new business opportunities by exposing business and application services for integration by other divisions, business organizations, or platforms. It has most comprehensive across platforms on the market, including the IBM eServer, IBM iSeries, and IBM zSeries.
- Management - XML based administrator client that works over HTTP. This allows the administrator to create and manage the cluster quickly and easily.
- Volume Maintenance - Support for Java Management Extensions (JMX), which records and logs statistics on usage and resources. This is the standard way to manage a J2EE environment and expose the WebSphere administrative data to partners like Tivoli and others for management integrations.
- Security - Security authentication options to include Kerberos tokens for strong authentication security for client/server applications. Also, provides open Security Programming Interfaces (SPIs) for integration into those third-party solutions.

3

Setting up the development environment

In this chapter, we discuss the following sections:

- Workbench fundamentals show the different perspectives, views, editors, and projects that are available with Workbench.
- This chapter introduces our sample application, which will be used throughout the redbook.

© Copyright IBM Corp. 2003. All rights reserved.

3.1 Workbench fundamentals

In Chapter 1, “What is Linux?” on page 1, we described Linux and its advantages. In Chapter 2, “WebSphere application development for Linux” on page 5, we introduced the foundation of WebSphere software platform and listed some of the new feature for WebSphere Studio Application Developer and Server Version 5. In this chapter we will show you the Workbench features that are available in the WebSphere software platform.

Workbench is an integrated development environment (IDE) that has flexible *perspectives*. These *perspectives* contain *views* and *editors* that provide a common way for members of the development team to create, manage, and navigate resources. For example, a Java developer would work most often in the *Java* perspective, while a Web designer would work in the *Web* perspective.

A *perspective* defines an initial set and layout of views and editors for performing a particular set of development activities. You can change the layout and the preferences and save the perspective that you have customized. One or more perspectives can be open in a single Workbench window. You can switch between perspectives in the same window, or you can open a new Workbench window.

We will briefly describe the different perspectives that are available in the Application Developer.

3.1.1 Resource perspective

This is the default perspective. This perspective is always shown at the top of the list when you select **Window -> Open Perspective** then **Open** from the menu. For a list of perspectives, select **Other** from the menu. Figure 3-1 shows the different perspectives that are available for your development.

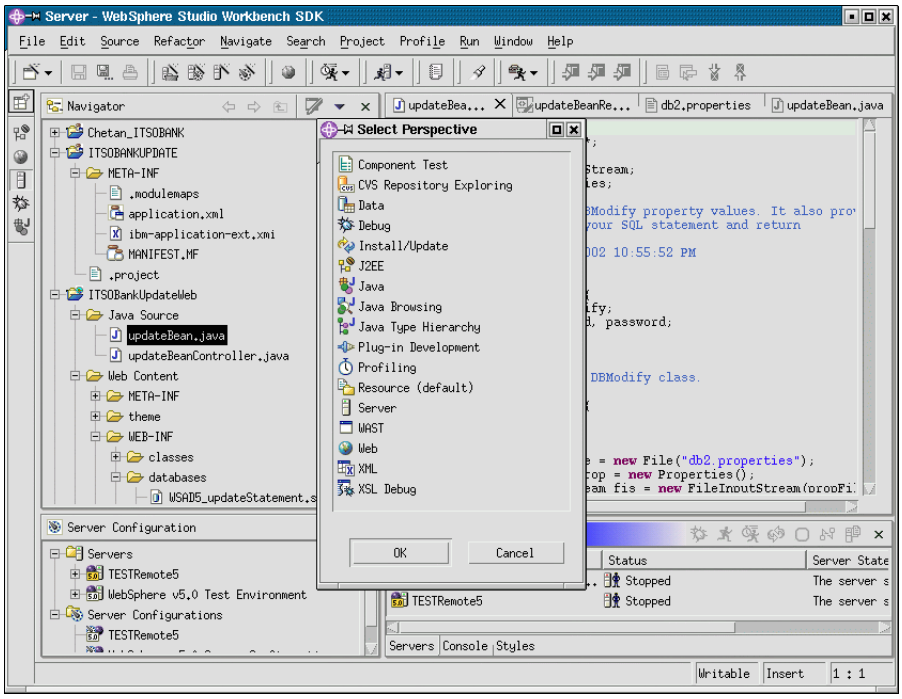


Figure 3-1 Select perspective

The Resource perspective contains the following (Figure 3-2).

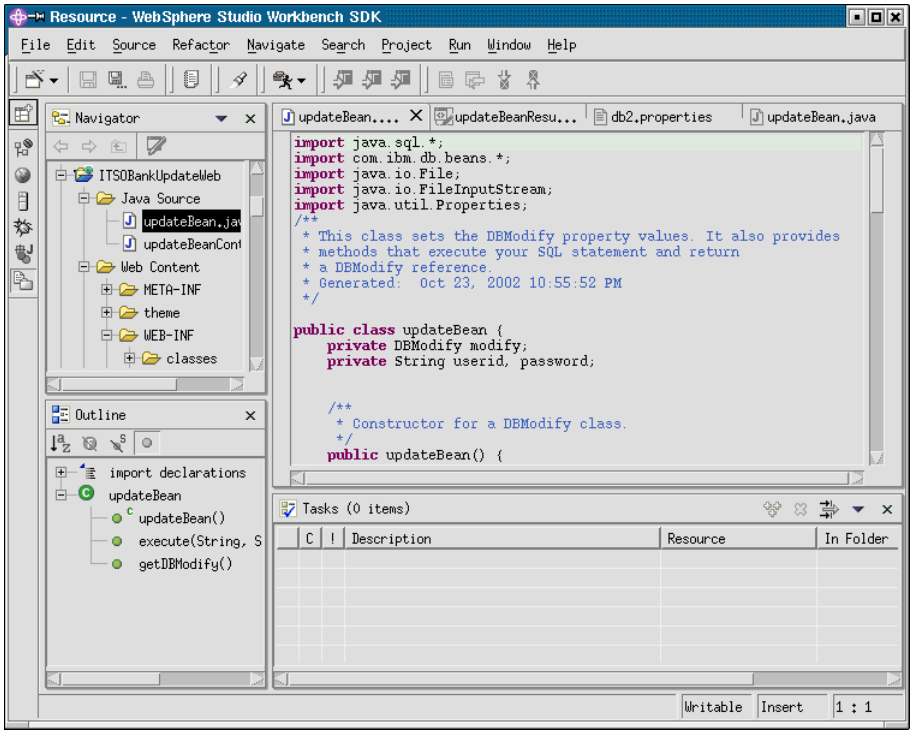


Figure 3-2 Resource perspective

This perspective contains three panes:

- Top left - Shows Navigator views
- Top right - Reserved for editors of the selected resources
- Bottom left - Shows Outline view of the resource opened in the active editor
- Bottom right - Shows Tasks and Console view

3.1.2 Java perspective

Java perspective is used to edit and build Java code. The Java perspective contains the following (Figure 3-3).

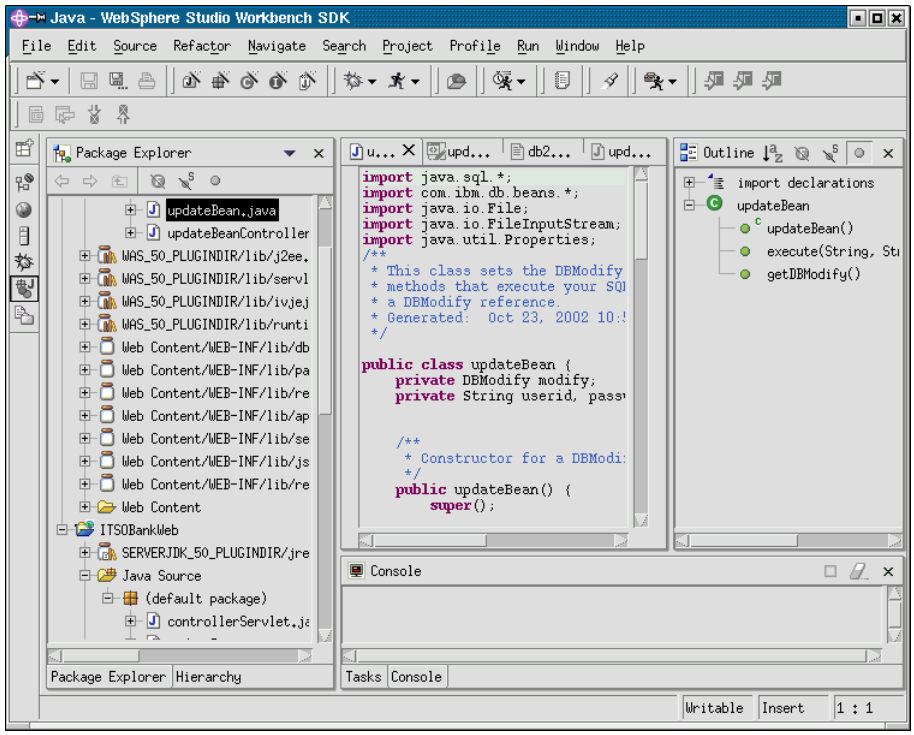


Figure 3-3 Java perspective

This perspective contains four panes:

- Left - Shows Packages and Hierarchy view
- Middle - Reserved for editors. Multiple files can be edited at one time.
- Right - Shows Outline view of the file currently in the active editor.
- Bottom - Shows the Task view for error messages and user tasks; the Search view for result of search operations; and the Console view for program output

3.1.3 Web perspective

In this perspective you can build and edit Web resources such as servlets, JSPs, HTML pages, deployment descriptor, web.xml, and images. The Web perspective contains the following (Figure 3-4).

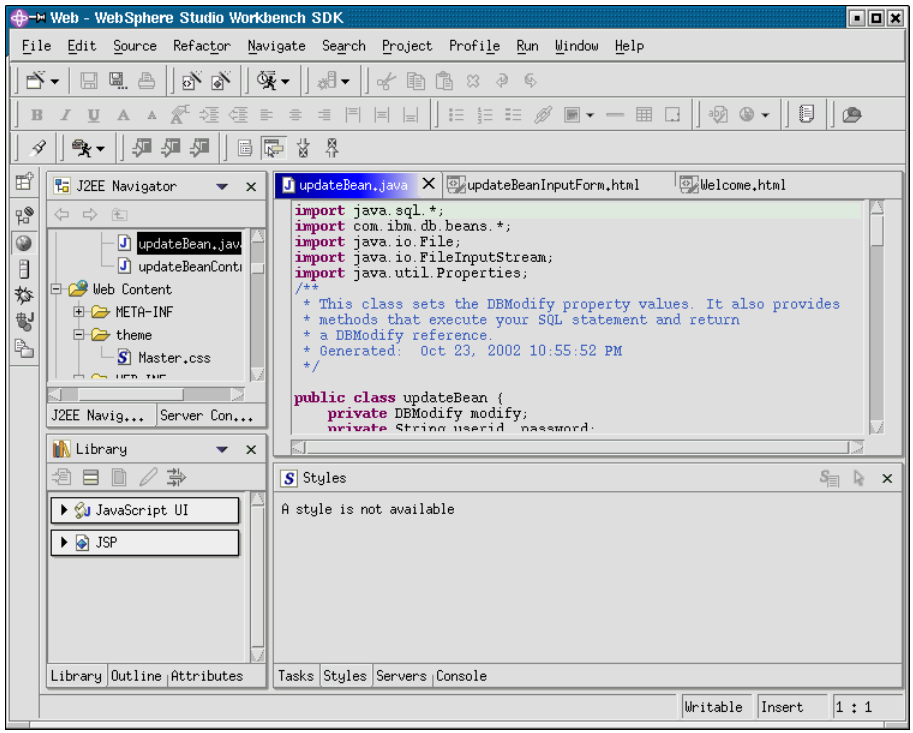


Figure 3-4 Web perspective

This perspective contains four panes:

- Top left - Shows the Navigator view, which displays the folders and files of the project
- Top right - Reserved for editors
- Bottom left - Shows the Outline view for the active editor or the Gallery for HTML and JSP files.
- Bottom right - Shows Tasks, Properties, Links, Thumbnails, Styles, Color and Palette views.

We will elaborate a little more on the deployment descriptor:

The Web application deployment descriptor is the *web.xml* file. This file contains information about the servlets and JSPs in your Web application. This is used to build a WAR file for a project, and contains the necessary information for deploying a Web application module.

3.1.4 J2EE perspective

In this perspective you can develop the EJBs and manage J2EE deployment descriptors (EARs). This view displays a logical view of the EJBs with their fields, keys, and underlying Java files. The J2EE perspective contains the following (Figure 3-5).

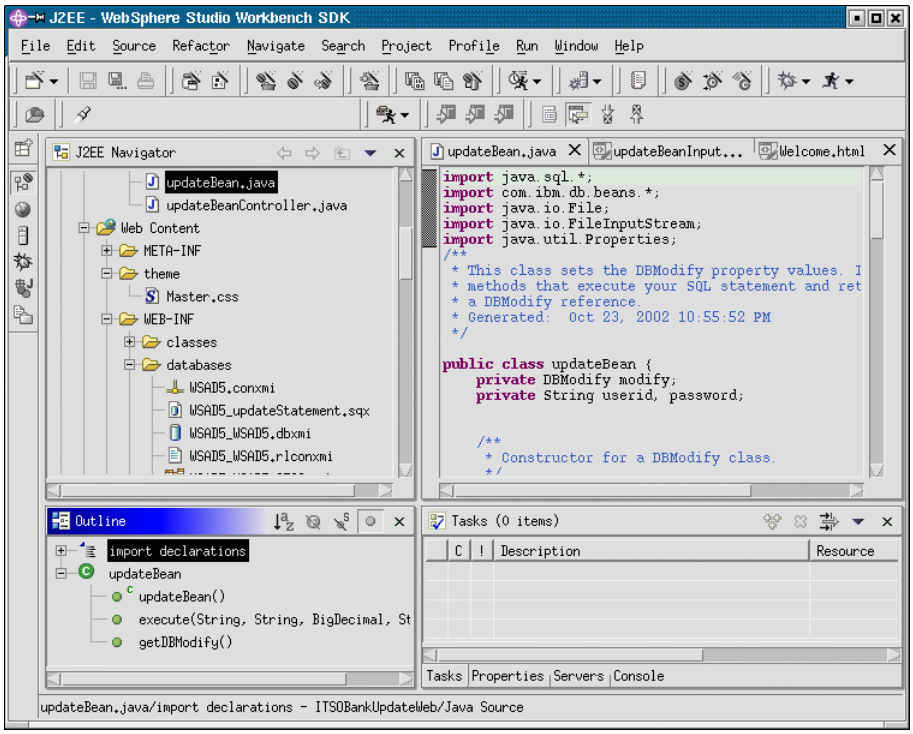


Figure 3-5 J2EE perspective

This perspective contains four panes:

- Top left - Shows the J2EE and Navigator views used to display the logical structure and resources of the project
- Top right - Reserved for editors
- Bottom left - Shows the Outline view for the active editor
- Bottom right - Shows Tasks view or Properties view of a select resource

3.1.5 Server perspective

Application Developer provides support for the local and remote test environments. In order to test a Web application, it has to be published to a

server by importing the EAR file of the project into the server. The local server runs inside the Application Developer. For the remote server, the IBM Agent Controller is used to start the remote server on another machine. Server can also be started in debug mode. In debug mode, breakpoints can be placed in your project files to help you find problems. The Server perspective contains the following (Figure 3-6).

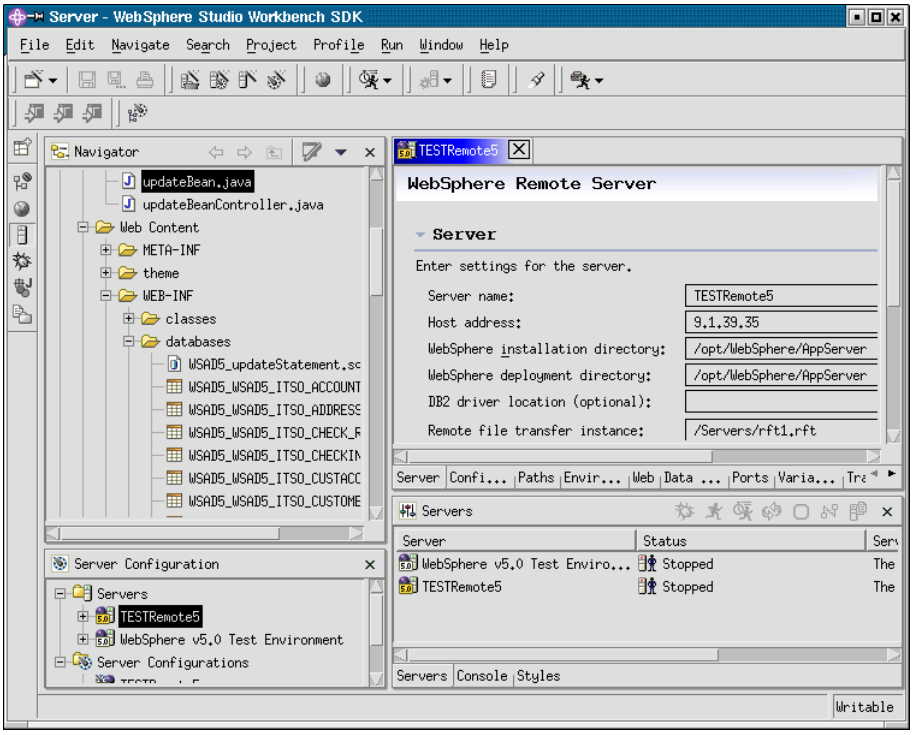


Figure 3-6 Server perspective

This perspective contains four panes:

- Top left - Shows the Navigator views used to display the logical structure and resources of the project
- Top right - Reserved for editors and browsers
- Bottom left - Shows the Server Configuration view with all defined server instances and their configurations
- Bottom right - Shows Server Control Panel, where the server can by placed on start, stop, debug and trace mode.XML

3.1.6 XML perspective

In this perspective the developer can build DTDs, XML schemas, XML, XSLT, and mapping between XML and different backend databases. The XML perspective contains the following (Figure 3-7).

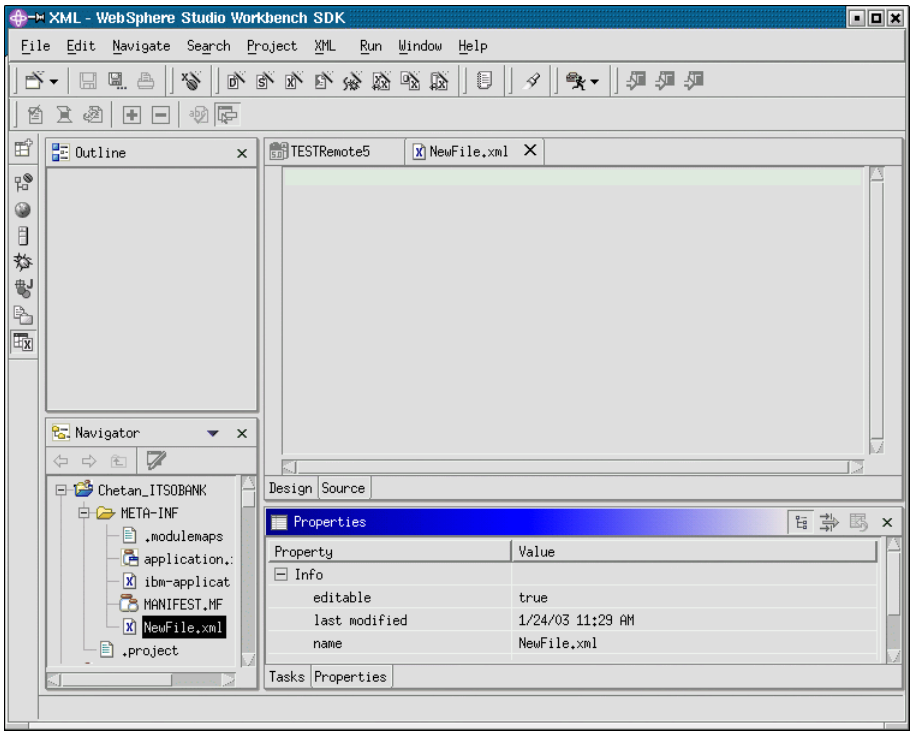


Figure 3-7 XML perspective

This perspective contains four panes:

- Top left - Shows the Outline view for the active editor
- Top right - Reserved for editors
- Bottom left - Shows the Navigator view that displays the folders and files of the project
- Bottom right - Shows the Tasks view that shows problems to be fixed and user defined tasks

3.1.7 Data perspective

From this perspective you can access the relational database tools. You can browse or import database schemas in the DB Explorer view. Editors and tools

are provided to create and manipulate local descriptors, generate DDL from local descriptors, and run existing DDL to create local descriptors. The Data perspective provides views and tools for definition and maintenance; or descriptor for database, schemas, and table definitions. The Data perspective contains the following (Figure 3-8).

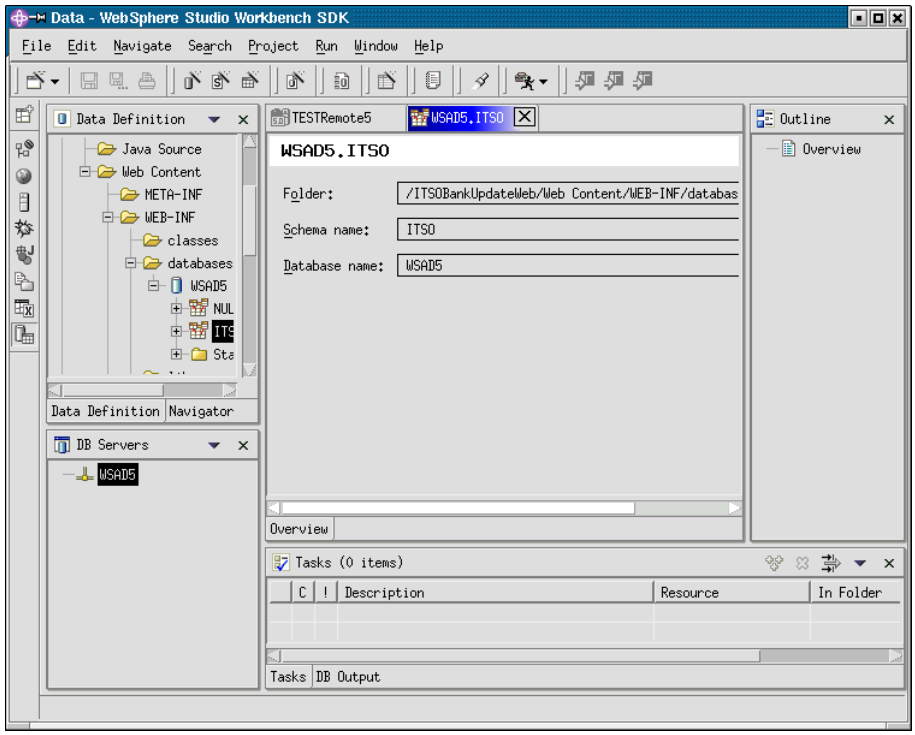


Figure 3-8 Data perspective

This perspective contains four panes:

- Top left - Shows DB Explorer, Data, and Navigator view
- Top middle - Reserved for editors
- Top right - Shows the Outline view
- Bottom - Show the Tasks view

3.1.8 Debug perspective

This perspective supports testing and debugging of your applications. The Debug perspective contains the following (Figure 3-9).

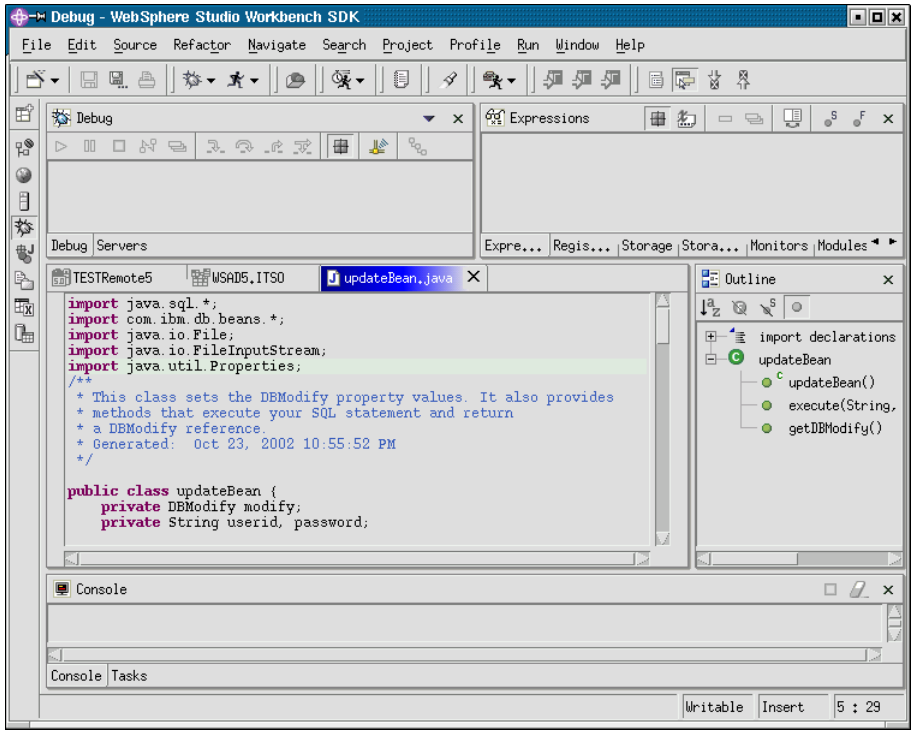


Figure 3-9 Debug perspective

This perspective contains five panes:

- Top left - Shows Servers and Debug views
- Top right - Shows Breakpoints, Inspector, Variables, and Display views
- Middle right- Shows the Outline view of the currently displayed source
- Middle left - Shows the Source view
- Bottom - Shows the Console

3.1.9 Profiling perspective

This perspective lets you test your application's performance early in the application development cycle. The profiling tools collect data related to the Java program's run-time behavior and present this data in graphical and non-graphical views. This is useful for performance analysis and understanding your Java programs. You can use this to view object creation and garbage collection, execution sequence, thread interaction, and object references. The Profiling perspective contains the following (Figure 3-10).

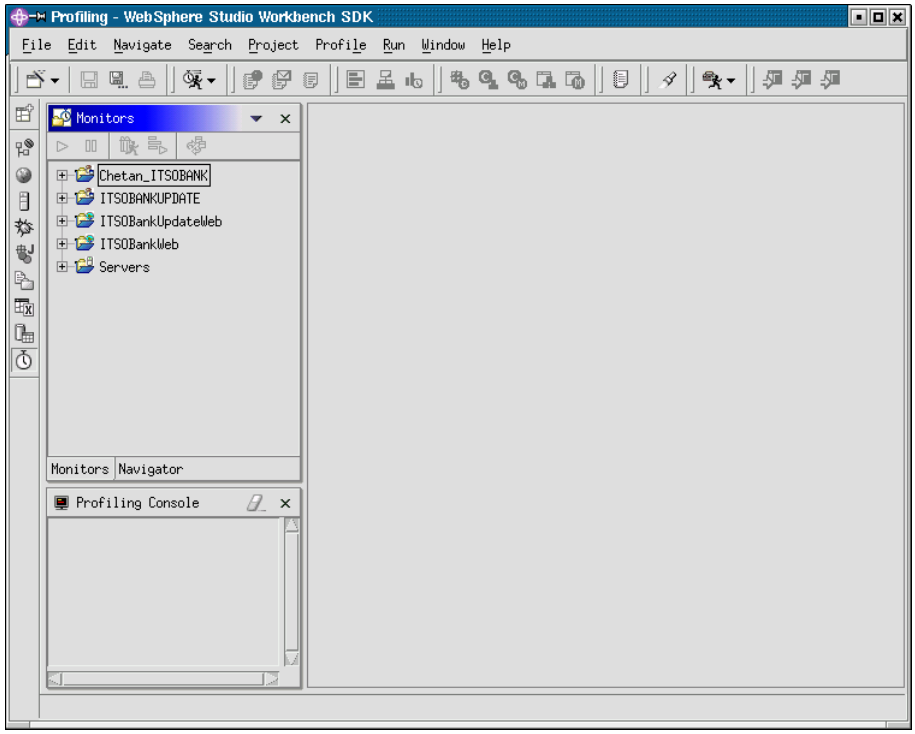


Figure 3-10 Profiling perspective

This perspective contains three panes:

- Top left - Shows the Monitors view, which contain the profiling resources
- Bottom left - Shows the Profiling Console view
- Right - Shows the profiling views such as heap, execution flow, object references, method execution and invocation, and class and method statistics

3.1.10 Team perspective

In this perspective different developers manage shared projects. The Team perspective contains the following (Figure 3-11).

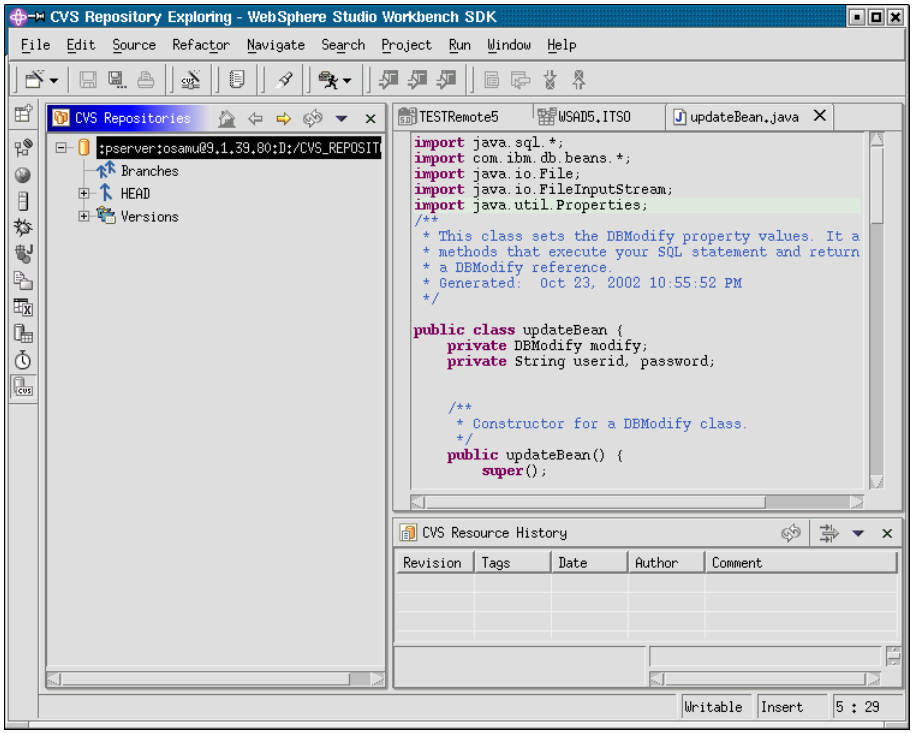


Figure 3-11 Team perspective

This perspective contains four panes:

- Top left - Shows the CVS Repositories view
- Top right - Shows the Source view
- Bottom right - Shows CVS Resource History view

3.1.11 Help perspective

In this perspective the developer has access to the online help. It lets the user access help documents.

To change the default perspective follow these steps:

1. Select **Window -> Preferences** from the menu bar.
2. Expand the **Workbench** item on the left and select the **Perspectives** preference page.
3. From the list of perspectives, select the one you want to define as the default.
4. Click **OK**.

Since you have a good understanding of these perspectives, we will now describe the *views* that are in most perspectives.

3.1.12 Workbench views

Views are ways of navigating through the information in your Workbench. The perspective usually determines the type of view that you will require. For example, the Java perspective includes the Packages view and the Hierarchy view to help you work with Java packages and hierarchies.

Here are the different views:

- Navigator - The most common view is the Navigator view. The Navigator panel shows you how the different resources are structured in different folders. These resources are files, folders, and projects.
- Outline - The Outline view gives you an overview of the key elements that make up the resource that is being edited. It allows quick and easy navigation through you resource. By selecting one of the elements in the Outline view, the line in the editor view that contains the selected element gets highlighted.
- Properties - When you click **a resource** in the Navigator view and then click the **Properties** tab at the bottom of the screen, you can view the different properties of that resource. The Properties view contains general things such as the full path on the file system, the date when it was last modified, and the size.
- Tasks - The Tasks view contains two types of elements: problems and tasks. Problems are tool determined issues that have to be resolved. For example, Java compile errors, or broken links for HTML/JSP files. They are automatically added to the Task view when working with the tool. When you double-click on a **problem**, the editor for the file containing the problem opens, and the cursor is pointed at the location of the problem. The tasks can be manually added. For example, you can add a task that reminds you that you have to implement a Java method. Place the cursor in the method's implementation, right-click, and select **Add -> Task**. When you double-click, the file opens and the cursor is located in the method. You can also add general tasks that do not refer to a specific file.

When you open a file, the software automatically opens the editor that is associated with the file. For example, an HTML editor is for .html, .htm and .jsp files.

3.1.13 Workbench projects

A project contains files and folders that are part of your application. In the Workbench, all folders and files must be contained in projects. Projects are used

for building, version management, sharing, testing, and deployment. You can create different types of projects in WebSphere Studio; for example, Web and Java projects.

We will briefly describe the different projects the are available in the Application Developer:

- Java - Java projects are associated with the Java builder that compiles Java source files automatically. Java projects can be exported as JAR files or into a directory structure.
- EAR - An Enterprise Application project is used to develop a J2EE enterprise application. An EAR project consists EJB modules, Web applications, and an application client.
- Web - Web projects contain the resources needed for Web applications, including servlets, JSPs, Java files, static documents (for example HTML pages or images), and any associated metadata. A Web project is deployed as a Web module (WAR file).
- EJB - EJB projects contain the resources for EJB applications. An EJB project is deployed as an EJB module (JAR file).
- Server - Server projects contain the necessary information to deploy application to an application server for testing.
- Application Client - Application client projects are deployed as a JAR file. They contain the resources needed for application client modules.
- Enterprise Application - Enterprise Application project contains the hierarchy of resources that are required to deploy an enterprise (J2EE) application. It contains a combination of Web modules, EJB modules, JAR files, and application client modules. It includes a deployment descriptor and an IBM extension document, as well as files that are common to all J2EE modules that are defined in the deployment descriptor. An Enterprise Application project is deployed in the form of an .ear file.

3.2 Sample application

In every chapter, the ITSO Bank example is used to show the various technologies and functions of WebSphere Application Developer and/or WebSphere Application Server, Version 5 for Linux.

The book is divided into four parts: the Introduction, Web application development, building and deploying Web applications, and the Appendix. The ITSO Bank example is used in the Web application development, testing, and the deploying of Web application parts.

The following is a list of topics that are covered in the *Web Application Development* section.

3.2.1 Web application using HTML, JSP, servlets, and JavaBeans

This chapter consists of three projects. The *ITSOBankSelectWeb* project follows the model in Figure 3-12.

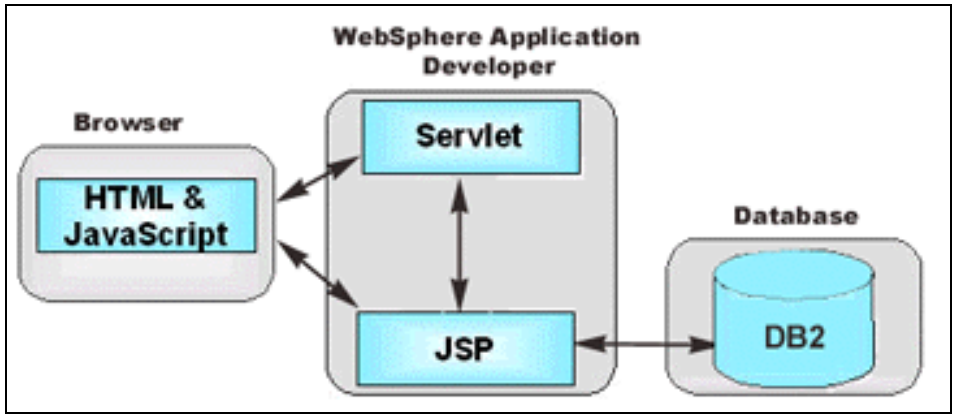


Figure 3-12 ITSOBankSelectWeb project model

In this model, we show how to use JSP, servlets, and the database.The customer will request account balance information from the database as they submit the user ID and password for their account.

The *ITSOBankUpdateWeb* project follows this model (Figure 3-13).

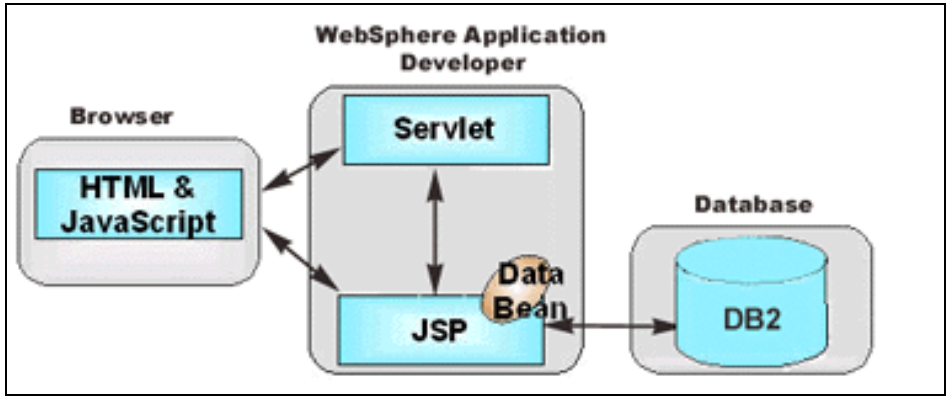


Figure 3-13 ITSOBankSelectUpdate project model

In this model, we show how to use JSP, servlet, JavaBeans, and database. The customer will be able to transfer the balance from one account to another. The customer will submit their account ID for their account(s).

Both of the above models are then combined to produce the *ITSOBankWeb* project. Through this example we show the various technologies that you can use for your projects.

3.2.2 Using Enterprise JavaBeans with your Web application

The ITSO Bank example follows this model (Figure 3-14).

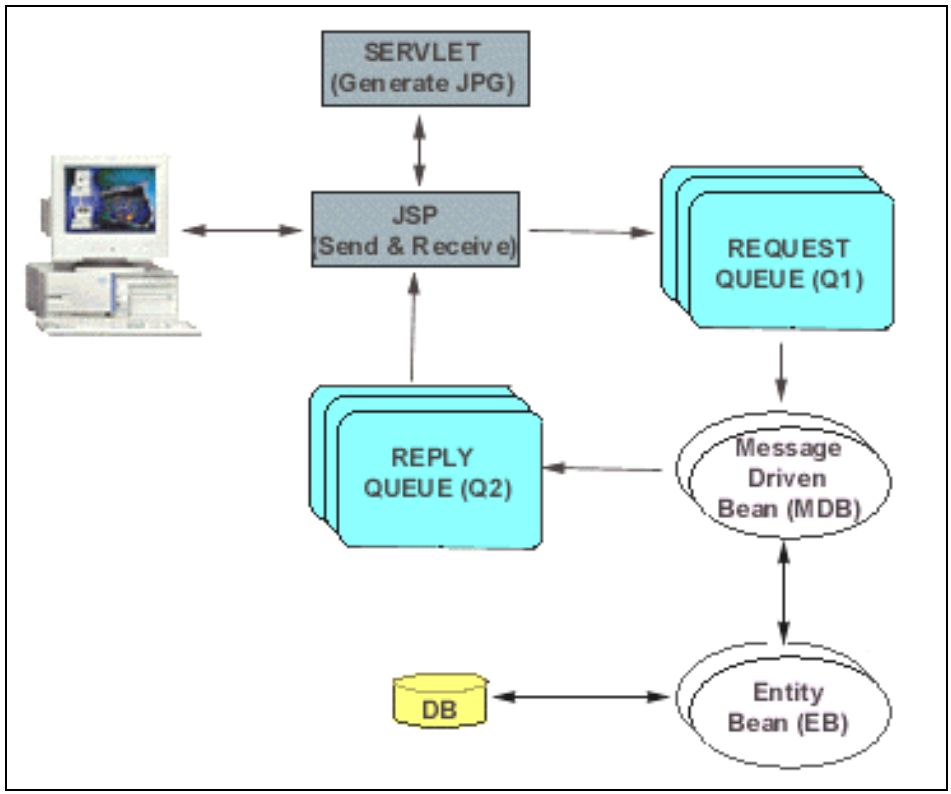


Figure 3-14 ITSOBankEJB project model

In the above model, the customer requests to re-order cheques. The customer will enter the customer ID and quantity. The request is then sent to a JSP program that consist of two main functions: sending and receiving. The sending function sends the request to the first queue (Q1). The queue then passes the necessary information to the Message Driven Bean (MDB). The Message Drive Bean sends the information to the Entity Bean (EB), which performs the

necessary SQL query to the backend database. Once the database is updated, a confirmation is send back to the EB. The EB sends this information to the MDB, which sends it to the second queue (Q2). The Q2 forwards the information to the JSP's receiving function. This function interacts with the servlet and produces an HTML output with a sample cheque.

Since this is an example, the JSP contains the receiving function. Normally, you would only send the data to the Q1 and not receive any confirmation.

3.2.3 Generating Web application using XML

The ITSO Bank example follows this model (Figure 3-15).

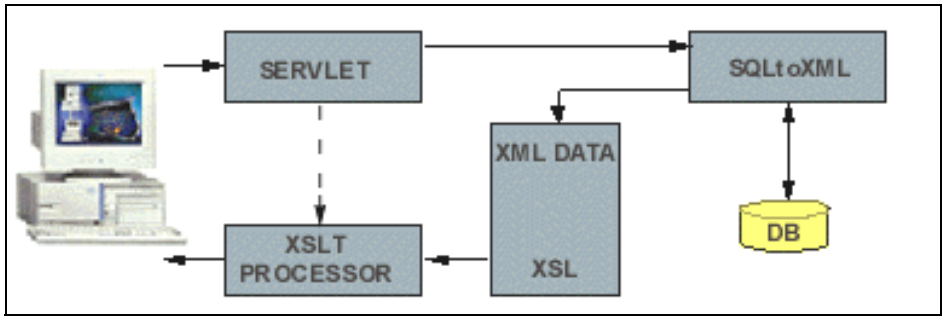


Figure 3-15 ITSOBankXML project model

In this model we show how you can dynamically generate XML from the SQL query at runtime using the SQLtoXML class library within a servlet. The customer submits a user ID and password to the servlet. The servlet calls the SQLtoXML class library and the specific information is selected from the database. The class library sends the XML document and combines it with the XSL stylesheet. This is processed by the XSLT processor and the customer's account balances are shown in simple HTML.

3.2.4 Testing and deploying Web application

Here are a list of topics that are covered in the testing and deploying Web application section:

Ant

We demonstrate the capabilities of Ant as a build tool using the ITSOBankEJB project, which is provided in the Enterprise JavaBeans chapter.

Deployment of applications

We demonstrate how to install, configure, and deploy an Enterprise Application. Also, we show you how to use the command line tools to administer your Enterprise Application. Since Ant can be used for deployment, we show you how to deploy the ITSOBankEJB project. Our Ant deployment process follows this model (Figure 3-16).

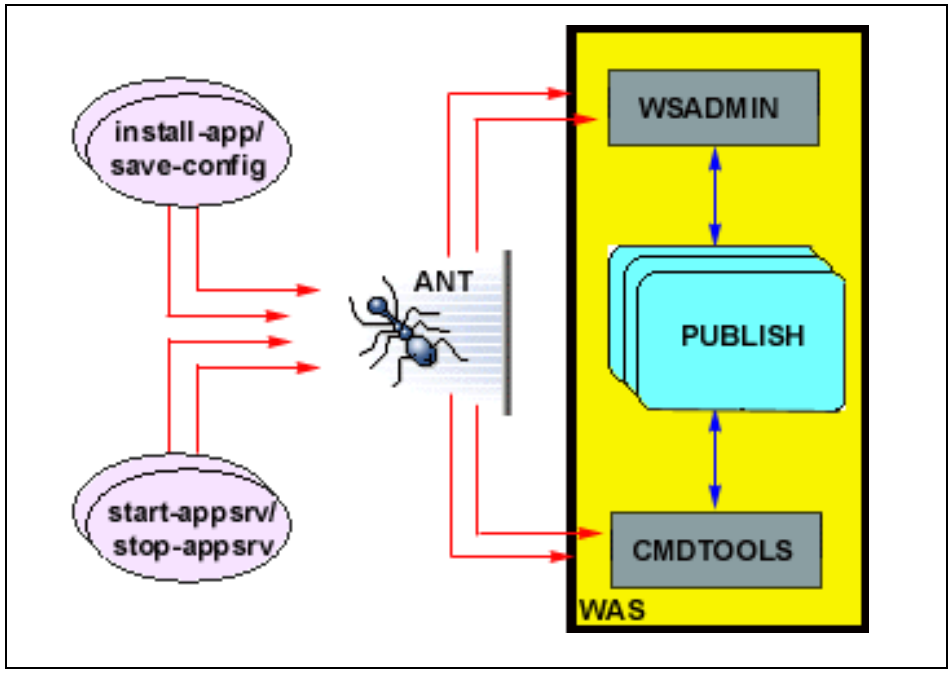


Figure 3-16 Deployment using Ant

3.2.5 Database design for ITSO Bank application

Lastly, the following is an overview of our database design. The ITSO Bank model consists of the following entities and relationships (Figure 3-17).

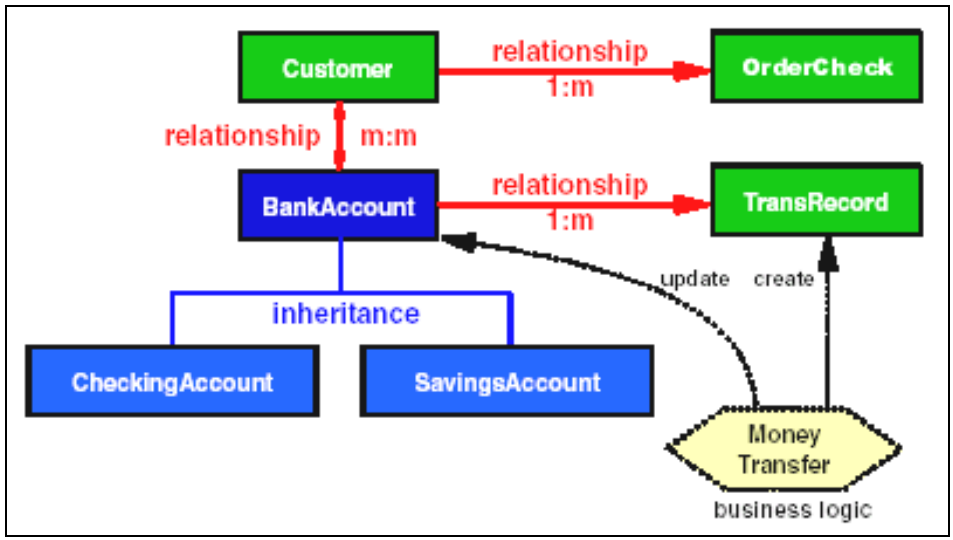


Figure 3-17 ITSOBankDatabase project model

The above model shows the entities and relationships in our database. The *Customer* entity has a one-to-many relationship with the OrderCheck entity. The *Customer* entity also has a many-to-many relationship with the BankAccount entity. The BankAccount entity has a one-to-many relationship with the TransRecord entity, and inherits the CheckingAccount and SavingsAccount entities. Using the various entities and relationships, we will implement various ITSO Banking examples to show how you can use technologies such as JavaBeans, JSP, EJB, Web Services, XML, and others.

4

HTML, JSP, servlet, JavaBeans, and database

In this chapter, we discuss the following sections:

- 4.1, “Preparing for development” on page 36 shows you how to start application development using WebSphere Application Developer
- 4.2, “HTML” on page 41 describes the components and uses of HTML using ITSOBankWeb project’s login page
- 4.3, “JSP” on page 43 introduces JSPs using the ITSOBankWeb project’s account information Web page
- 4.4, “Servlet” on page 46 introduces servlets using the ITSOBankWeb project’s controller servlet program, and gives an overview of all the components that are part of the deployment descriptor
- 4.5, “JavaBeans” on page 50 introduces JavaBeans using the ITSOBankWeb project’s update bean program and transaction confirmation Web page
- 4.6, “Database” on page 53 describes the ITSO Bank database in details, and how to connect to the database and build a SQL query within WebSphere Studio Application Developer

© Copyright IBM Corp. 2003. All rights reserved.

4.1 Preparing for development

In this chapter, we have used three Web applications: *ITSOBankSelectWeb*, *ITSOBankUpdatWeb,* and *ITSOBankWeb*. We show you how to build Web applications using HTML, JSP, servlets, and JavaBeans as they communicate with the IBM DB2 Database.

Installing the database and the source code for each project can be found in the Appendix.

The following is a brief description of each project:

- *ITSOBankSelectWeb* - Queries the database to retrieve the customer's information from the customer and address tables. The customer provides the user ID and password to retrieve their records. This project is built using the *Database Web Pages* wizard. The wizard creates an HTML login form, a JSP result's page, and a servlet. The project follows this model (Figure 4-1).

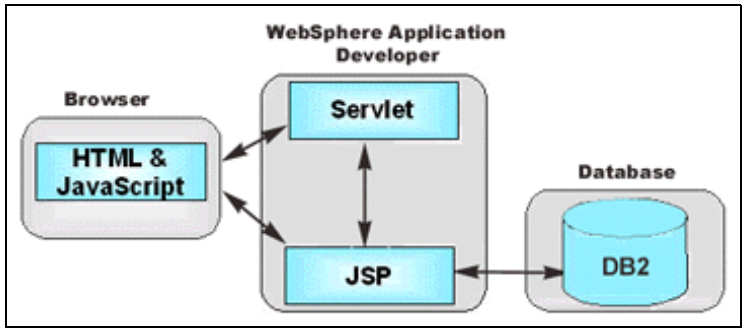


Figure 4-1 ITSOBankSelectWeb model

- *ITSOBankUpdateWeb* - Queries the database to update the customer's bank account table given the account ID and balance. This project is built using the JavaBean Web Pages wizard. The wizard creates an HTML login form, a JSP result's page, a servlet, and a JavaBean. The project follows this model (Figure 4-2).

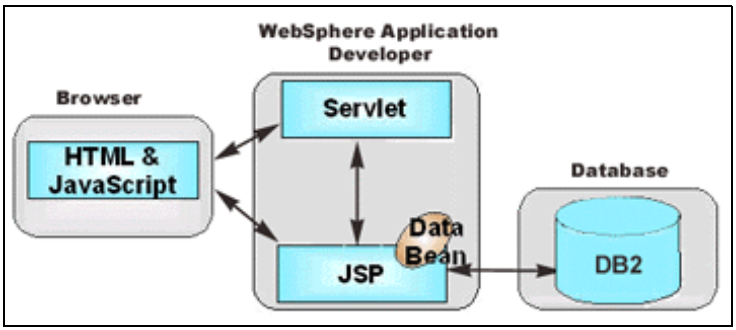


Figure 4-2 ITSOBankUpdateWeb model

- *ITSOBankWeb* - This project combines both of the above models to build a banking example that lets the customer login, check account balances, and transfer money between accounts.

4.1.1 Creating a new project

In this section, we begin by showing you how to start application development using WebSphere Application Developer. To describe each step we have used the **ITSOBankSelectWeb** project:

1. Start the WebSphere Studio Application Developer.
2. Select **File -> New -> Project** to create a new project.
3. Select **Web** then select a **Web project**. The Web project contains resources related to a Web application.
4. Specify a project name `ITSOBankSelectWeb`.
5. Select **Use default** as the default directory.
6. You can select to build a Static or Java2 Enterprise Edition (J2EE) Web project. Static Web projects use HTML with JavaScript and graphics. J2EE Web projects use technologies that are served with the J2EE Application Server. Choose **J2EE** Web project.
7. The **Web Project Features** contain enhancements that you can add to your Web project. This will be added to your WEB-INF directory. Select <**Include Tag Libraries for database access and accessing JSP object** > in the Web Project features list, then click **Next**.
8. Select **New Enterprise application** project.
9. Specify the project name as `ITSOBankSelectEAR`.
10. Select **Use default** as the default directory.
11. Do not modify the context root.

12. Select **J2EE level 1.3**. This will use the servlet 2.3 and JSP 1.2 specifications that are used in WebSphere Application Server Version 5.0, then click **Finish**.

After completing the above steps you can switch to the Resource perspective to see your project directories. In the Navigator window you will see two new directories: ITSOBankSelectEAR and ITSOBankSelectWeb (Figure 4-3). We will describe the contents of these directories.

- ITSOBankSelectEAR (Figure 4-3).

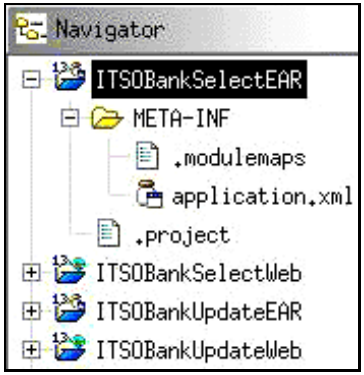


Figure 4-3 ITSOBankSelectEar directory

In this directory the most important file is application.xml. This is the Deployment descriptor for the enterprise applications that are responsible for associating Web and EJB projects to a specific EAR file.

- ITSOBankSelectWeb:

 This directory contains the following (Figure 4-4):

 - *Java Source* directory: contains the servlets or your Java programs
 - *classes* directory: contains compiled class files
 - *lib* directory: contains JAR files used by Web applications
 - *Master.css* file: a stylesheet for HTML and JSP
 - *web.xml* file: the Deployment descriptor, which contains general information about the Web application, servlet mappings, and security information. The *ibm-web-bnd.xmi* file is used for WebSphere bindings. The *ibm-web-ext.xmi* file is used for IBM-specific extensions.

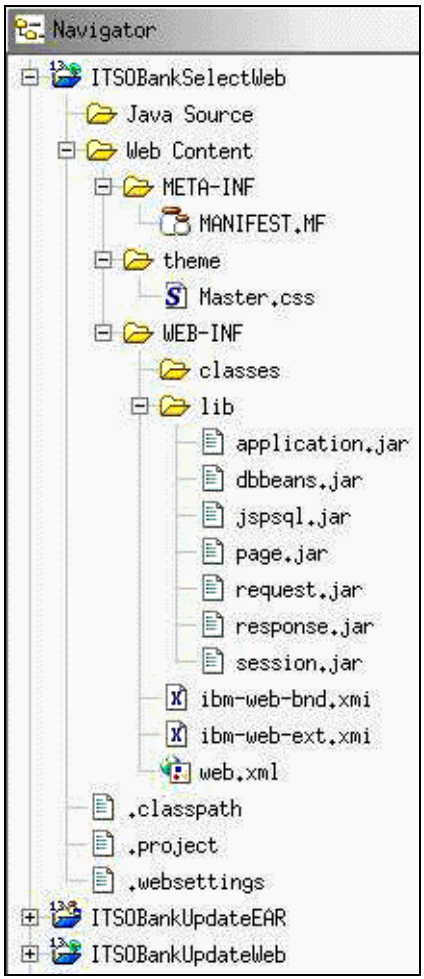


Figure 4-4 ITSOBankSelectWeb directory

Now switch to the Data perspective to continue creating your Web project.

- Create a connection to the database and the select statement that will be used for your project. This is shown in the database section of this chapter. Turn to this section to complete the step.

Switch to the Resource perspective. You will notice that the database folder is populated with the necessary database connection and your SQL statement:

1. Select the **ITSOBankSelectWeb** directory. Click **File ->New -> Other**.
2. Select **Web** then the **Database Web Pages** wizard. Click **Next**. This wizard will build a framework to create Web pages from a SQL query.

3. Your destination folder will be /ITSOBankSelectWeb/Web Content.
4. You can choose a Java package or use the default. We will use the default.
5. Select the **Select Statement** from the SQL Statement Type window.
6. Select the model as **IBM Database Access Tag Library - Select Statement**. The other models include the master details pattern or JavaBeans. The details page displays more details of the selected data. The JavaBeans model is to include a JavaBean program in your Web project. To explore this model, step through the ITSOBankUpdateWeb project. Click **Next** to continue.
7. In this window, we will select **Use Existing SQL Statement**, browse to the /ITSOBankSelectWeb/Web Content/WEB-INF/database/WSAD5/Statements directory, and select **selectStatement**. Click **Next** to continue.
8. In the Runtime Connection Page, the fields will already include the database connection parameters. Click **Next** to continue.
9. In the Controller Page, ensure that **Create a new Front Controller** is selected. This will build a servlet for your project. Click **Next** to continue.
10. For the next three database connection windows click **Next** and then finally **Finish** to complete your Web project.

Assuming that the connection to the database and the select statement have been created correctly, the wizard will create the necessary HTML, JSP, and servlets. Click **Run on Server** to test the connections before making any changes to your Web project.

The following is a summary of the steps that we used to complete the ITSOBankSelectWeb project:

1. In the Resource perspective, create a Web project.
2. In the Data perspective, create a database connection, and an SQL statement.
3. In the Resource perspective, use the Database Web Pages wizard to build a framework, and connect the various components of your Web project.

The following is a summary of the steps that we have used to build the ITSOBankUpdateWeb project:

1. In the Resource perspective, create a Web project.
2. In the Data perspective, create a database connection, an SQL statement, and select to generate JavaBeans.
3. In the Resource perspective, use the JavaBeans Pages wizard to build a framework, and connect the various components of the Web project.

The import function in WebSphere Studio Application Developer can help you import our Enterprise Application projects to your workspace.

In the next sections we will go through the different components of the ITSOBankWeb project. Using each component, we will explore the use of various technologies.

4.2 HTML

In this section, we will be using parts of the ITSOBankWeb login page and the login.html to explain the basics of HTML.

HTML stands for Hypertext Markup Language. HTML is used to provide a simple markup language to create hypertext documents that are portable across platforms. The main components of HTML are the meta information, and HTML head and body. The following is a brief description of what they are:

- **Meta information**

Example 4-1 Meta tags in HTML

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
<HTML>
<META http-equiv="Content-Type" content="text/html">
<META name="GENERATOR" content="IBM WebSphere Studio">
```

In this component, you can show the version of the document, type of document, and the date the document was created or updated.

- **HTML header**

Example 4-2 Header tags in HTML

```
<HEAD>
<LINK href="theme/Master.css" rel="stylesheet" type="text/css">
<TITLE>ITSO Banking Example</TITLE>
</HEAD>
```

In this component, you will usually see the title or a link to the stylesheet. The stylesheet helps towards the presentation of the Web site. In the example above, the link element is an external link to style sheet. You can also import a stylesheet using the *@import* and a *style* attribute on an element inside the HTML body.

- **HTML body**

Example 4-3 Body tags in HTML - ITSOBankSelectWeb login form

```
<BODY>
<H1><IMG border="0" src="images/header.gif" width="523" height="75"></H1>
<!--Java Script-->
<SCRIPT LANGUAGE="JavaScript" TYPE="text/javascript">
<!--
function submitForm(nav){
   document.loginForm.elements["command"].value = nav
   document.loginForm.submit()
}
//-->
</SCRIPT>

<Form name="loginForm" method="post" action="controllerServlet">
User ID: <INPUT type="text" name="userid" maxlength="8"><BR>
Password: <INPUT type="password" name="password" maxlength="8">
<P>
<INPUT TYPE="hidden" NAME="command" VALUE="accountInfo">
<INPUT TYPE="hidden" NAME="new_input" VALUE="true">
<A href="javascript:submitForm('accountInfo')">Submit</A>
</Form>
<P><IMG border="0" src="images/footer.gif" width="523" height="20"></P>
</Body></HTML>
```

In this component, the document consists of information and links to other documents.

Combining various components allows you to create HTML documents. After you have generated the HTML document, you can view it in a browser. The login.html page found in under Appendix 4.6.1, “ITSO Bank database” on page 53 will look as follows (Figure 4-5).

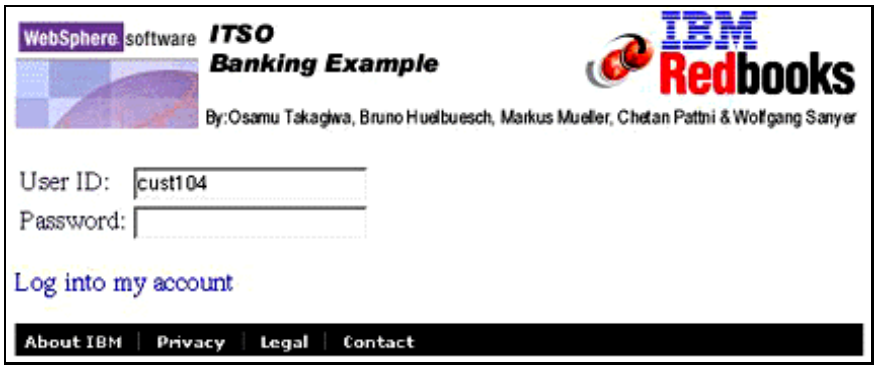


Figure 4-5 ITSOBankWeb login page

4.3 JSP

In this section, we will be using the ITSOBankWeb project and accountInfo.jsp to explain the basics of JSP and the use of SQL tag library, which are utilized by the Database Web Pages Web wizard. The accountUpdate.jsp uses similar library functions, hence, we will not elaborate on this JSP. A simple example of the wizard can be found in the ITSOBankSelectWeb project.

JSPs are JavaServer Pages. They are portable across any platform. These are programs that run in the front and middle layer, acting as Web interface for requests that are coming from a user or HTTP clients and database. In JSP you can add dynamic content using expressions. The tag <% and %> and an = sign at the start of the sequence makes a Java *expression*. This *expression* is evaluated at runtime. This allows the JSP to generate dynamic HTML. JSPs also allow you to write Java code. The syntax is the same as the expression, but without the = sign. This Java code is know as a *scriptlet*. The *accountInfo.jsp* uses expressions and scriptlets. We will briefly describe how the taglib tags, session variables and connecting, query, and display database information is used in accountInfo.jsp:

- **The taglib tags**

Example 4-4 taglib tag in JSPs

```
<%@ taglib uri="/WEB-INF/lib/jspsql.jar" prefix="dab_cust_account" %>
```

To extend the functionality of JSP, you can include directives within your JSPs. There are page, taglib, and include directives. The page directive is explained in the JavaBeans section. The include directive is used to include the content of another file, and the taglib library tells the JSP compiler where to find

specified library tags. The taglib contains two parameters, uri and prefix. The uri specifies where to look for the tag library description and the prefix is a reference name for the specified tag library. In the above example, we use the jspsql.jar tag library to access the JSP SQL tags.

- **Session variables**

Example 4-5 Session variables in JSPs

```
<% if(request.getParameter("new_input") != null){
   if(request.getParameter("userid") != null){
      session.setAttribute("userid", request.getParameter("userid"));
   }
   if(request.getParameter("password") != null){
      session.setAttribute("password", request.getParameter("password"));
   } } %>

<% String inputUSERID = (String) session.getAttribute("userid");
String inputPASSWORD = (String) session.getAttribute("password");
%>
```

There are many defined variables, requests, responses, outs, and others. Session is one of the defined variables that is used to simplify the JSP code. This is part of the HttpSession object that is associated with the request. In the above code, we have used the request.getParameter to acquire the parameter that is usually the user's responses. Using the session.set Attribute, we have assigned it to a variable and later set it to a string variable by using session.getAttribute. The string variable can be further manipulated within the JSP. You can also define the session attribute to true or false in the page directive. By default the session value is true.

- **Database connection, query, and display tags**

Example 4-6 database connection, query and display tags in JSPs

```
<%-- Connect to the database --%>
<%! com.ibm.db.beans.DBConnectionSpec dsSpec = null; %>
<%
if (dsSpec == null) {
%>
<dab_cust_info:driverManagerSpec id="databaseConnection" scope="page"
   userid='<%=config.getInitParameter("username")%>'
   password='<%=config.getInitParameter("password")%>'
   driver='<%=config.getInitParameter("driverName")%>'
   url='<%=config.getInitParameter("url")%>'/>
<%
   dsSpec = databaseConnection;
 }
%>
```

```
<%--Execute the query--%>
<dab_cust_info:select id="select_cust_info" scope="request"
connectionSpecRef="<%=dsSpec%>">
 <dab_cust_info:sql>
 SELECT
   ITSO.CUSTOMER.FIRSTNAME,
   ITSO.CUSTOMER.LASTNAME,
......
<TABLE border="0">
   <TBODY>
<TR>
<dab_cust_info:repeat name="select_cust_info" index="rowNum" over="rows">
          <TD><b>Name: </b><dab_cust_info:getColumn index="1"/>
<dab_cust_info:getColumn index="2"/></TD></TR><TR>
          <TD><b>Address: </b><dab_cust_info:getColumn
index="3"/><dab_cust_info:getColumn index="4"/></TD>
</dab_cust_info:repeat>
       </TR>

.....
```

Since we have used the *Database Web Pages* wizard, the connection to the database and the SQL query is defined within the JSP. In the above lines of code, the config.getInitParameter acquires the values for the various parameters from the Web application configuration file, web.xml. The *<...:driverManagerSpec>* tag creates a DBConnectionSpec object that has the required information to make a connection to the database. The reference to this object is stored in a static variable called dsSpec. To execute the query, the information is collected, then a connection is established, and the specified SQL statement is executed within the *<..:sql>* tag. The data is added to the DBSelect cache. The next section uses a *<...:repeat>* tag to get the data and display as HTML. We have used the IBM customer JSP SQL tags, which allow you to access data in a database using JSP tags. They provide a function similar to the TSX tags that are part of WebSphere.

Figure 4-6 is to show you what the output will look like once you have successfully logged into your account, the accountInfo.jsp page.

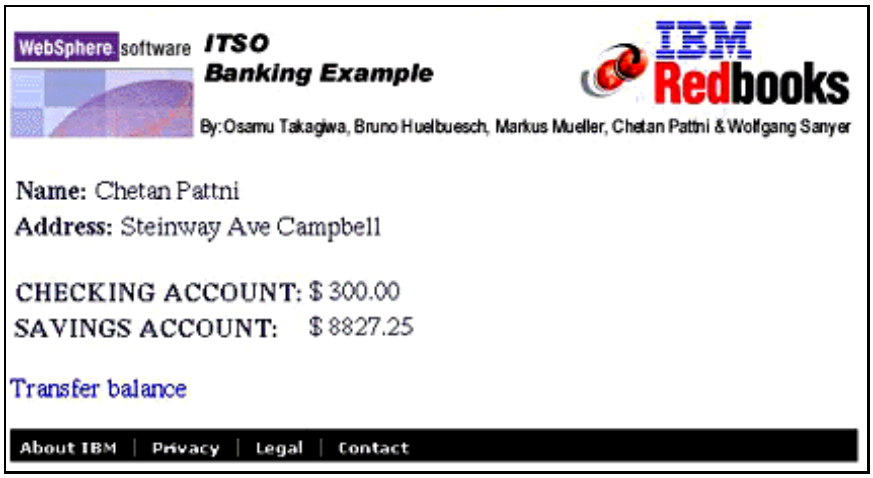


Figure 4-6 ITSOBankWeb account information page

Once you click **Transfer balance**, the accountUpdate.jsp page will look like Figure 4-7.

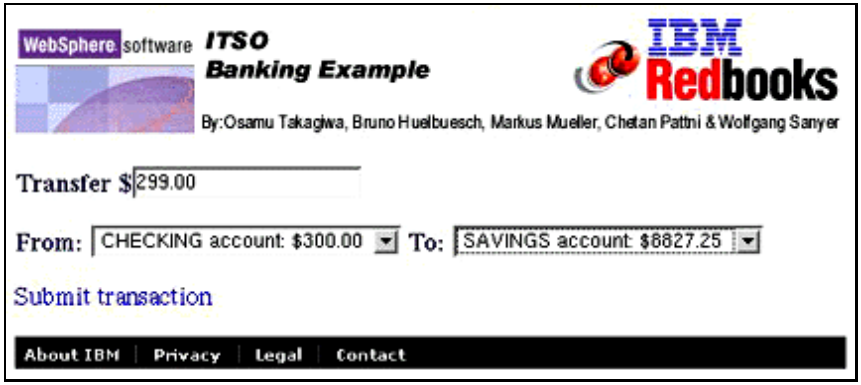


Figure 4-7 ITSOBankWeb account update page

4.4 Servlet

In this section, we will be using the **ITSOBankWeb** servlet, the controller Servlet.java to explain the basics of servlets, and also, web.xml, the Web Deployment Descriptor.

Servlets are Java programs that are portable across platform. These are programs that run in the middle layer acting as controller for requests that come

from a Web browser, or HTTP clients, and database. The main components of servlets are the init, doGet, doPost, performTask, performServices, and dispatch methods. The following is a brief description of what each one looks like.

The init method

This method is used to load the initial variables. In this case the values reside in the Deployment Descriptor under the servlet tag. You can also specify other variables that are required for the servlet to initialize.

Example 4-7 init method in servlets

```
public void init(ServletConfig config) throws ServletException {
   super.init(config);
}
```

The doGet method

When the user submits an HTML form, it specifies if the method is going to be HTTP get or post. If the form uses the get method, the doGet method should be used. The doGet method takes two arguments, HttpServletRequest and HttpServletResponse, where the first handles incoming data, and the later handles outgoing data. In our example we override the method with the performTask method.

Example 4-8 doGet method in servlets

```
public void doGet(HttpServletRequest req, HttpServletResponse resp)
   throws ServletException, IOException {
   performTask(req, resp);
}
```

The doPost method

If the form uses the post method, the doPost method should be used. The doPost method takes two arguments, HttpServletRequest and HttpServletResponse, where the first handles incoming data, and the later handles outgoing data. This is exactly the same as its counter part doGet. In our example we override the method with the performTask method.

Example 4-9 doPost method in servlets

```
public void doPost(HttpServletRequest req, HttpServletResponse resp)
   throws ServletException, IOException {
   performTask(req, resp);
}
```

The performServices

The performServices method is used to perform user authentication. In our example, we perform this using our JSP. The servlet is used as a gateway to pass user requests and responses from the front to the backend, and then finally back to the user.

Example 4-10 performServices method in servlets

```
public void performServices(HttpServletRequest request,
   HttpServletResponse response) {
}
```

The performTask

In our example, the performTask method processes input variables, but before it can do this, it ensures that performServices method does not have any services to be completed for the user. The next step is to get the nextPage variable that will determine what the servlet is going to launch next, or what it will do through the dispatch method.

Example 4-11 performTask method in servlets

```
public void performTask(HttpServletRequest req,
   HttpServletResponse resp)
   throws ServletException, IOException {
   String nextPage;
   try {
      performServices(req, resp);
      nextPage = getInitParameter(req.getParameter("command"));
   } catch (Exception ex) {
      nextPage = getInitParameter("error_page");
   }
   dispatch(req, resp, nextPage);
}
```

The dispatch method

The dispatch method passes the input variables using the dispatch.forward function. Then using the nextPage variable launches the next page in response to the user's request.

Example 4-12 dispatch method in servlets

```
public void dispatch(
      HttpServletRequest req, HttpServletResponse resp, String nextPage)
      throws ServletException, IOException {
      RequestDispatcher dispatch = req.getRequestDispatcher(nextPage);
      dispatch.forward(req, resp);
```

```
        }
```

Servlets main function is to read any data sent by users, generate, and format the results; and send data back to the users. Servlets are efficient, portable, and powerful over the traditional Common Gateway Interface and CGI-like technologies. They are efficient because each request is a lightweight Java thread instead of a operating system process. Since servlets are written in Java, they are portable. Servlets are supported by every major Web server. They are also part of the Java 2 Platform. Overall, servlets become powerful and easy-to-use, since they are more reliable and reusable than any other programming language.

The Web Deployment Descriptor contains general information about the Web application, servlet mappings, and security information. The main components of web.xml are the web-app, display-name, servlet, servlet-mapping and taglib tags.

The web-app tag

In this tag, you define the name of your Web application.

Example 4-13 web-app tag in Web deployment descriptor

```
<web-app id="WebApp">....</web-app>
```

The display-name tag

In this tag, you define the name of your Web application as it appears in the WebSphere Application Developer.

Example 4-14 Display-name tag in Web deployment descriptor

```
<display-name>ITSOBankSelectWeb</display-name>
```

The servlet tag

In this tag, the servlet's actual name, display name, and its initial parameters are defined. The initial parameter in our example contains the DB2 connection information that is used by the servlet or JSP to connect to the database.

Example 4-15 Servlet tag in Web deployment descriptor

```
<servlet><servlet-name>accountInfo</servlet-name>accountInfo<display-name><
/display-name><jsp-file>/accountInfo.jsp</jsp-file><init-param><param-name>
username</param-name><param-value>db2inst1</param-value></servlet>
```

The servlet-mapping tag

In this tag, the servlet path is defined for servlets that are used by your servlet or JSP.

Example 4-16 Servlet-mapping tag in Web deployment descriptor

```
<servlet-mapping><servlet-name>accountInfo</servlet-name><url-pattern>
/accountinfo.jsp</url-pattern></servlet-mapping>
```

The taglib tag

In this tag, the library path is defined for libraries that are used by your servlet or JSP.

Example 4-17 taglib tag in Web deployment descriptor

```
<taglib><taglib-uri>jspsql</taglib-uri><taglib-location>
/WEB-INF/bin/jspsql.jar</taglib-location></taglib>
```

4.5 JavaBeans

In this section, we will be using the ITSOBankWeb, confirmUpdate.jsp, and updateBean.java to explain the basics of JavaBeans, and the use of the SQL tag library that is utilized by the JavaBean Web Pages wizard. A more simpler example of the wizard can be found in the ITSOBankUpdate Web project.

The confirmUpdate.jsp has three main components: A class import, Jsp useBean tag, and an execute function. The following is a brief description of what each one looks like.

A class import statement

The page directive is used to import a list of packages. In our case we will import the updateBean class.

Example 4-18 Class import statement in JSPs

```
<%@ page
import="updateBean" contentType="text/html; charset=WINDOWS-1252"
pageEncoding="WINDOWS-1252"
%>
```

The Jsp useBean tag

This defines *<..:useBean>* tag that has an *id* and the *class*. The *id* is used in within the JSP to refer to the class file. This statement lets you load a bean in the

JSP page, and then instantiate an object of the class by binding it to the specified variable.

Example 4-19 useBean tag in JSPs

```
<jsp:useBean id="updateDBBean" scope="request" class="UpdateDB"
type="UpdateDB"/>
```

An execute function

This function is used to pass the *newBalance*, *formacctID,* and *toacctID* variables, which are passed to the JSP as hidden fields and the user's response.

Example 4-20 Execute function in JSPs

```
<%
updateDBBean.execute(new java.lang.Float
(request.getParameter("newBalance")), new
java.lang.String(request.getParameter("fromacctID")), new
java.lang.String(request.getParameter("toacctID")));

%>
```

Since we have used the *Java Bean Web Pages* wizard, the connection to the database and the SQL query is defined within the JavaBeans. The JavaBeans program, *updateBean.java*, has two main methods: the initializer and execute. The following is a brief description of what each one looks like.

The initializer

In this method the JavaBean is initialized. The database driver and the URL are defined. Also, the SQL command is defined with the necessary variables that will be replaced with actual values given by the execute method.

Example 4-21 initializer method in JavaBeans

```
protected void initializer() {
       modify1 = new DBModify();
       modify2 = new DBModify();
       try {
          modify1.setDriverName("COM.ibm.db2.jdbc.app.DB2Driver");
          modify1.setUrl("jdbc:db2:WSAD5");

          modify1.setCommand(
             "UPDATE ITSO.ACCOUNT SET BALANCE = :newBalance WHERE
ITSO.ACCOUNT.ACCID = :fromacctID");
          DBParameterMetaData parmMetaData = modify1.getParameterMetaData();
          parmMetaData.setParameter(
             1,
```

```
            "newBalance",
            java.sql.DatabaseMetaData.procedureColumnIn,
            java.sql.Types.CHAR,
            String.class);
         parmMetaData.setParameter(
            2,
            "fromacctID",
            java.sql.DatabaseMetaData.procedureColumnIn,
            java.sql.Types.CHAR,
            String.class);
....
```

The execute

This method gets the necessary input from the JSP. It then perform basic arithmetic on the account balance and calls the execute method to update the database with the new information.

Example 4-22 Execute method in JavaBeans

```
public void execute(Float newBalance, String fromacctID, String toacctID)
      throws SQLException {
      try {
         modify1.setUsername("db2inst1");
         modify1.setPassword("osamurs1");

         StringTokenizer st1 = new StringTokenizer(fromacctID);
         String fromAcctid = st1.nextToken();
         String prevfromBalance = st1.nextToken();

         float fromBalance = (Float.valueOf(prevfromBalance).floatValue())
- newBalance.floatValue();
         String frBal = Float.toString(fromBalance);

         modify1.setParameter("newBalance", frBal);
         modify1.setParameter("fromacctID", fromAcctid);
         modify1.execute();
......
```

Combining both of these components allows you to create a complete JavaBean program. The transaction will be processed by updateBean.java and the confirmUpdate.jsp page found in the Appendix 4.6.1, “ITSO Bank database” on page 53 and will look like Figure 4-8.

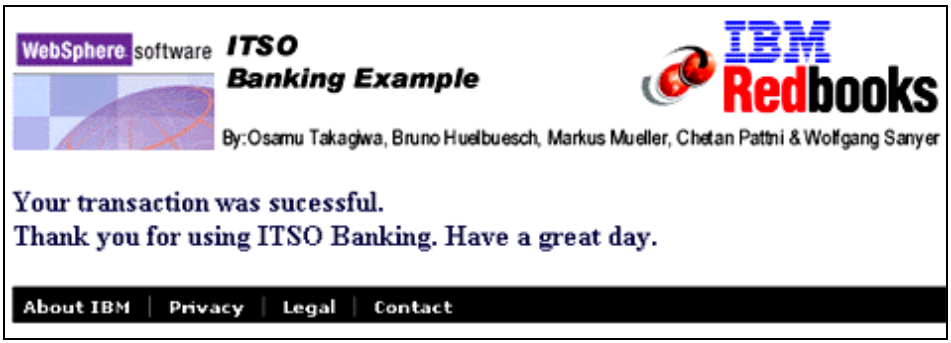


Figure 4-8 ITSOBankWeb confirmUpdate page

4.6 Database

In this section, we describe the ITSO Bank database in details, how to connect to the database, and how to build an SQL query within WebSphere Studio Application Developer. To complete the section you will need to have our database in your DB2 environment. Refer to the readme file located under the ITSO database directory of the sample code.

4.6.1 ITSO Bank database

As mentioned in the previous chapter, the WSAD5 database consists of entities and relationships to build a model. We will describe the relational database. The *WSAD5* database consists of the following tables:

- Customer table - The customer table contains the following information.

Table 4-1 Customer table

Column name	Type - length	Key	Nulls	Description
CUSTOMERI	INTEGER - N/A	PK	NO	Customer ID
TITLE	CHAR - 3	-	NO	Title
FIRSTNAME	VARCHAR - 30	-	NO	First name
LASTNAME	VARCHAR - 30	-	NO	Last name
USERID	CHAR - 8	-	YES	User ID
PASSWORD	CHAR - 8	-	YES	Password
PHONE	VARCHAR - 20	-	YES	Phone Number

- Address table - The address table contains following information.

Table 4-2 Address table

Column name	Type - length	Key	Nulls	Description
CUSTOMERID	INTEGER - N/A	PK,FK	NO	Customer ID
STREET	CHAR - 20	-	YES	Street number, name
CITY	CHAR - 12	-	YES	City
STATE	CHAR - 12	-	YES	State or country
ZIPCODE	CHAR - 10	-	YES	Postal code

- Customer Account table - The customer account table contains following information.

Table 4-3 Customer account table

Column name	Type - Length	Key	Nulls	Description
CUSTOMERID	INTEGER - N/A	PK,FK	NO	Customer ID
ACCID	CHAR - 8	PK,FK	NO	Account ID

- Account table - The account table contains following information.

Table 4-4 Account table

Column name	Type - length	Key	Nulls	Description
ACCID	CHAR -8	PK	NO	Account ID
BALANCE	DEC - (8,2)	-	NO	Balance
ACCTYPE	VARCHAR - 8	-	NO	Account type

- Re-order Check table - The re-order checks table contains the following information.

Table 4-5 Check_Reorder table

Column name	Type - length	Key	Nulls	Description
CK_RO_ID	INTEGER - N/A	PK	YES	Customer ReOrder ID
FK_CUST	INTEGER - N/A	FK	YES	Customer ID
QUANTITY	INTEGER - N/A	-	YES	Quantity

Column name	Type - length	Key	Nulls	Description
STATUS	VARCHAR - 15	-	YES	Status of order
DATE_ORDERED	TIMESTAMP - N/A	-	YES	Date, time ordered

We have not described the remaining tables since they are not part of any examples in this redbook. They are part of the database model to give you an incentive to expand on our example and utilize the database. Figure 4-9 is a overview of the tables in the database.

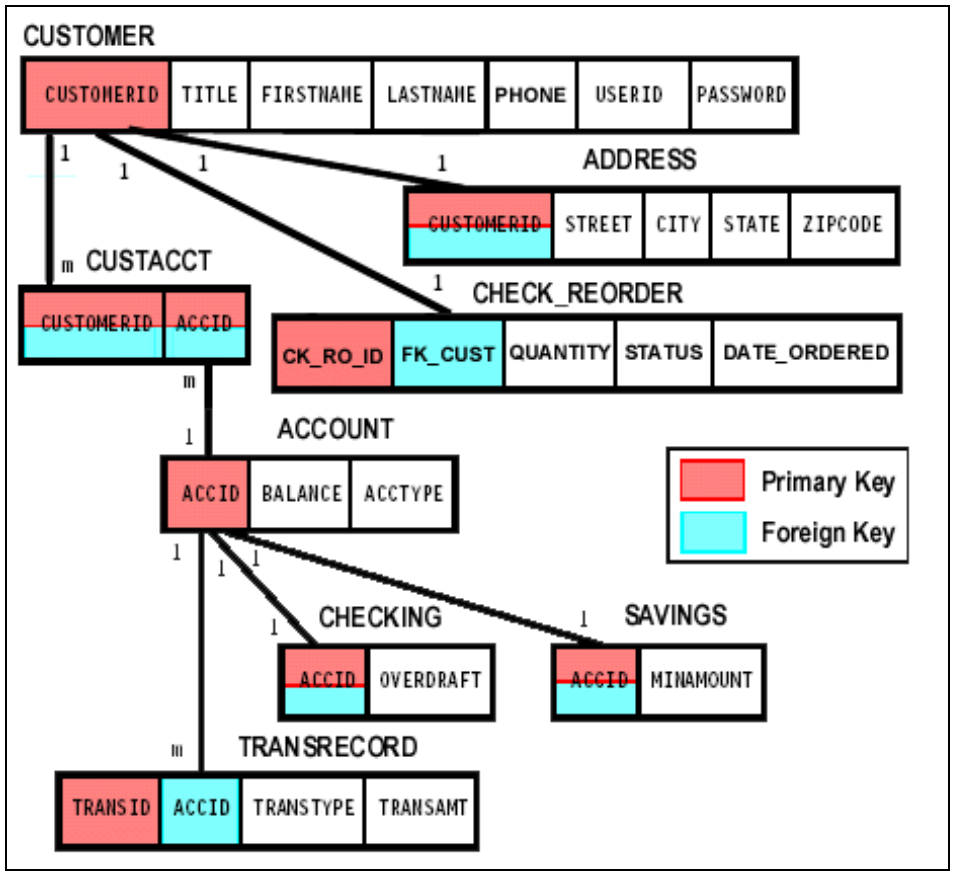


Figure 4-9 ITSO Bank example database

4.6.2 Connecting to a database from Application Developer

In this section, we will show how to connect to the database and import the connection to the project in the Data perspective:

1. Switch to the Data perspective.
2. In the DB Servers window right-click and select **New Connections**.
3. As your connection name, use ITSOBankWeb Connection or WSAD5.
4. Database name is WSAD5.
5. Enter the user ID and password for your database.
6. Use the IBM DB2 App Driver for a local database connection. Make sure that the class location is pointing to the db2java.zip in the /home/db2inst1/sqllib/java directory.
7. Click **Finish**.

Figure 4-10 is the directory structure of the database connection.

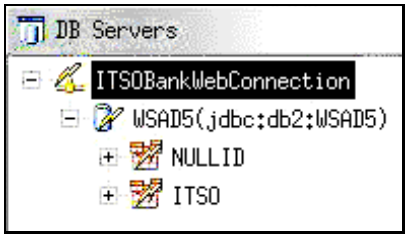


Figure 4-10 ITSOBankWebConnection directory

Follow these steps to import this database to the ITSOBankSelectWeb project:

1. Right-click **WSAD5(jdbc:db2:WSAD5)**.
2. Select **Import To Folder**. In the Import window browse to the Web Content directory in ITSOBankSelectWeb project.
3. Click **OK** then **Finish** to import.

A connection to the database will now exist in the WEB-INF directory for your project.

4.6.3 Using SQL Query Builder in Application Developer

In this section, we will show how to create a SQL statement for the ITSOBankSelectWeb project:

1. In the Data perspective, select **Create A New SQL Statement** from the task bar.
2. Select **SELECT** as the SQL statement.
3. Since you have imported the database to this project, click **Browse** to use existing database model.

4. Specify the SQL statement name as `selectStatement`. Click **Next**.
5. In the Tables tab, expand the **ITSO** folder and select **ITSO.CUSTOMER** and **ITSO.ADDRESS**.
6. Click the **greater-then arrow** button to transfer the selected table to the right window.
7. In the Columns tab, expand the selected tables and choose **TITLE**, **FIRSTNAME, LASTNAME, STREET, CITY, STATE,** and **ZIPCODE**.
8. Click the **greater-then arrow** button to transfer the selected columns to the right window.
9. In the Joins tab, join the **CUSTOMERID** from the Customer table to the **CUSTOMERID** in the Address table, as shown in Figure 4-11.

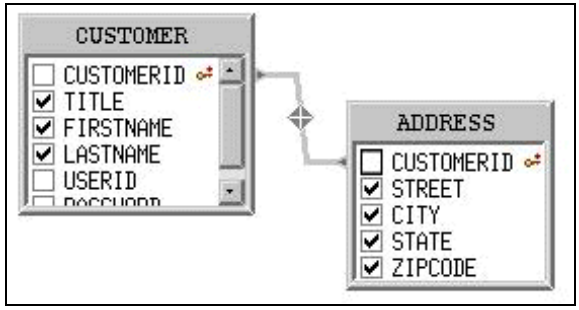


Figure 4-11 SQL joins window

10. In the Conditions tab, select **ITSO.CUSTOMER.USERID** in the Column cell.
11. Select the **equals** sign as the Operator cell and **Build Expression** for the Value cell.
12. In the Expression Builder window, select **Constant - numeric, string** or **host** variable. Click **Next**.
13. In the Expression Builder window, select Constant type as String constant. Click **Next**.
14. Use userid as the userid variable. Select **on Host** variable name. Click **Finish**.
15. In the And/Or cell select **AND.**
16. Follow the above steps for ITSO.CUSTOMER.PASSWORD. Click **Next**.

The SQL statement will look as follows.

Example 4-23 ITSOBankSelectWeb selectStatement

```
SELECT
ITSO.CUSTOMER.TITLE,
ITSO.CUSTOMER.FIRSTNAME,
ITSO.CUSTOMER.LASTNAME,
ITSO.ADDRESS.STREET,
ITSO.ADDRESS.CITY,
ITSO.ADDRESS.STATE,
ITSO.ADDRESS.ZIPCODE
   FROM
ITSO.CUSTOMER,
ITSO.ADDRESS
   WHERE
ITSO.CUSTOMER.CUSTOMERID = ITSO.ADDRESS.CUSTOMERID AND
ITSO.CUSTOMER.USERID = :userid AND
ITSO.CUSTOMER.PASSWORD = :password
```

17. Click **Execute** to test, and then click **Finish**. In the database folder in the Web project WSAD5_selectStatement.sqx contains the SQL query.

You can similarly create other SELECT, UPDATE, or other SQL statements that are required for your project.

5

Enterprise JavaBeans 2.0

This chapter provides the reader with an opportunity to learn about Enterprise JavaBeans application development, and deployment and testing with WebSphere Studio Application Developer Version 5.0. We will provide the ITSO Bank sample using EJBs, and we will walk through the operations necessary to implement business logic very quickly on a Web application server running on a Linux workstation or on a mainframe environment.

Before we start to develop an EJB, we propose to refresh your mind with a short description of the major components found in the EJB Version 2.0 specification.

© Copyright IBM Corp. 2003. All rights reserved.

5.1 The types of Enterprise JavaBeans

With EJB Version 2.0 there are now three types of Enterprise JavaBeans:

- **Message-driven Beans (MDBs)**. A MDB is a stateless Java component activated by messages that are mostly processed in asynchronous mode. According to the message paradigm, we distinguish between two messaging domains:
 - **Point-To-Point Messaging (PTP)**. This messaging domain is analogous to a mailbox, where the receiver decides when to open the message queue.
 - **Publish/Subscriber Messaging (PUB/SUB)**. This messaging domain allows publishing one message for a specific topic to many receivers. Many clients, who subscribed to a message topic, can receive the message.
- **Session beans**. A session bean is a Java server-sided component representing business logic or managing a process or a control flow. Session beans come in two modes:
 - Stateless session beans
 - Stateful session beans
- **Entity Beans**. An Entity Bean is a Java server-sided component that represents a business object and its associated data. Entity Beans are divided by their form of persistence:
 - Bean-Managed Persistence (BMP). The bean itself is responsible for implementing object persistence.
 - Container-Managed Persistence (CMP). The container, which is the runtime environment for all beans, performs the persistence on behalf of the bean.

5.1.1 Java Message-driven Beans

A Message-driven Bean is a new type of enterprise bean defined in the EJB 2.0 specification as an asynchronous message consumer. The bean is invoked by the container as the result of the arrival of a Java Message Service (JMS) message. The listener manager passes the message via a listener port to the bean. Message-driven Beans are stateless, server-sided, and not distributed components. They do not have EJBObject and EJBHome references. They are invoked by the container as soon as an asynchronous messages is delivered from JMS. Once the message arrived on the queue, the EJB cannot wait to receive a JMS message.

To a client, a Message-driven Bean is a JMS consumer that implements some business logic running on the server. A client accesses a Message-driven Bean by sending messages to the JMS destination for which the bean class is the MessageListener (Figure 5-1). The destination is either a Queue or a Topic.

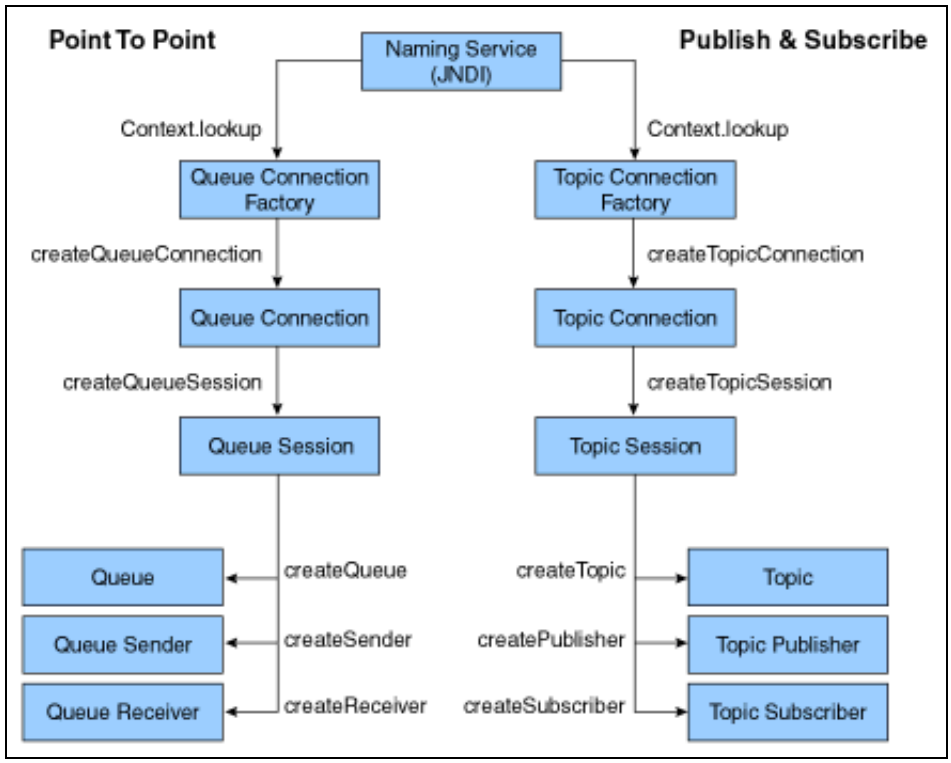


Figure 5-1 JMS class hierarchy

Message-driven Beans subscribe to specific message destination, which is a ListenerPort mapped to a queue or topic name. The ability to concurrently process messages makes Message-driven Beans extremely powerful enterprise beans. An EJB container can process thousands of messages from various applications concurrently by leveraging the number of bean instances. There is no guarantee about the exact order delivered to the bean instances. The bean should be prepared to handle messages that are out of sequence.

Configuring Embedded JMS Provider

It is assumed that you have a WebSphere Studio Application Developer V5.0 test server up and running. Before we can test JMS and Message-driven Beans with WebSphere Studio Application Developer Version 5.0, your test server requires

some configuration: JMS Provider activation, listener port name definitions, and queue name definitions for your test server.

Activate MQJD Provider implementation

The JMS provider for WebSphere Studio Application Developer V5.0 running on Linux is defined in the implfactory.properties file stored in the /opt/IBM/WebSphereStudio/eclipse/runtimes/base_5/properties/ directory. We recommend that you activate the MQJD JMS Provider entry.

Example 5-1 Activate MQJD JMS Provider in implfactory.properties

```
#Embedded JMS Provider
#com.ibm.ws.messaging.JMSProvider=com.ibm.ws.messaging.JMSEmbeddedProviderImpl

#MQJD JMS Provider
com.ibm.ws.messaging.JMSProvider=com.ibm.ws.messaging.JMSMQJDProviderImpl
```

If you have to change the file for activating the MQJD JMS Provider entry you have to logon as root (su).

Enable the Admin client

The message listener service requires a listener port, so that a Message-driven Bean can be associated with the listener port to retrieve messages from the JMS queue or topic. The definition of the listener port can only done via the WebSphere administrative console, which has to be enabled in the WebSphere V5.0 server configuration:

1. Switch to the Server perspective and select the **WebSphere v5.0 Test Environment**. (**Servers ->WebSphere v5.0 Test Environment**). Right-click **Open**.
2. Select the **Configuration notebook** tab (Figure 5-2) and enable the check box: **Enable administration client**. The Ports Notebook tab will show the Admin host port: 9090.
3. Press Ctrl+S to save changes and close the editor using Ctrl+F4.
4. Start WebSphere Studio Application Developer:
 a. If your V5.0 test server has already started, then restart the Web server. To restart, select the **WebSphere v5.0 Test Environment** in the server's view, and right-click **Restart**.
 b. If your server is not started, then right-click **Start** from the pop-up menu.
5. After the server starts, check the WebSphere console log in the server's Console view. You should find following messages: `ApplicationMg A WSVR0221I: Application started: adminconsole`

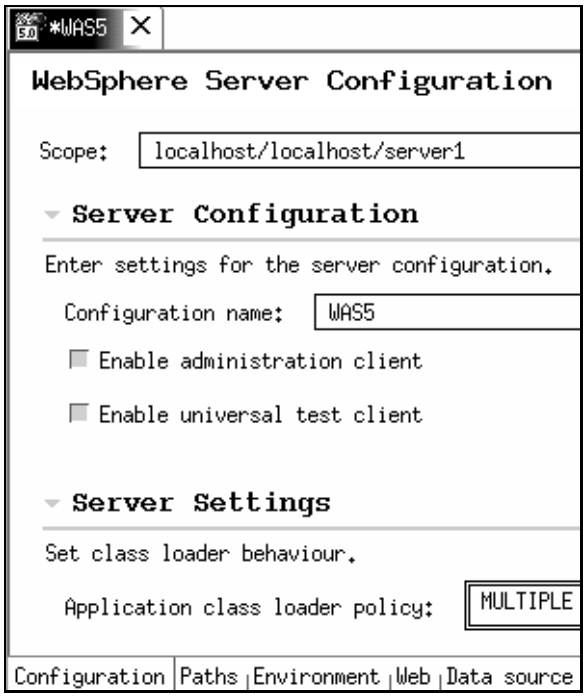


Figure 5-2 Enable the administration client

Set up Listener Port, Queue Names, and JNDI Mapping

Admin client can be started from the Servers view. Go to the Servers view then click the **Servers** tab if it is not selected. Right-click the **WebSphere AStudio Application Developer V5.0 server,** which has been started, and select **Run administrative client** from the popup menu. Your browser will start with the URL: `http://localhost:9090/admin/` If you use WebSphere Application Server, start the Admin client. For login, type a user ID and click the **OK** button**.**

To add a new listener port complete the following steps:

1. In the Navigation pane, select **Servers -> Application Servers.** This displays the properties of the application server in the content pane. Select **your-app-server** from the Application Server's list.
2. In the Additional Properties table, select **Message Listener Service**. This displays the Message Listener Service properties in the content pane.
3. In the Content pane, select **Listener Ports**. This displays a list of the listener ports.
4. In the Content pane, click **New**.

5. Specify appropriate properties for the listener port (Figure 5-3). Type in all required fields:
 a. Name (LP0)
 b. Initial State (Started)
 c. Connection factory JNDI name (jms/itso/QCF)
 d. Destination JNDI name (jms/itso/Q0)
6. Click **OK**.
7. To save your configuration, click **Save** on the task bar of the administrative console window. In the application pane you must click **Save** again.

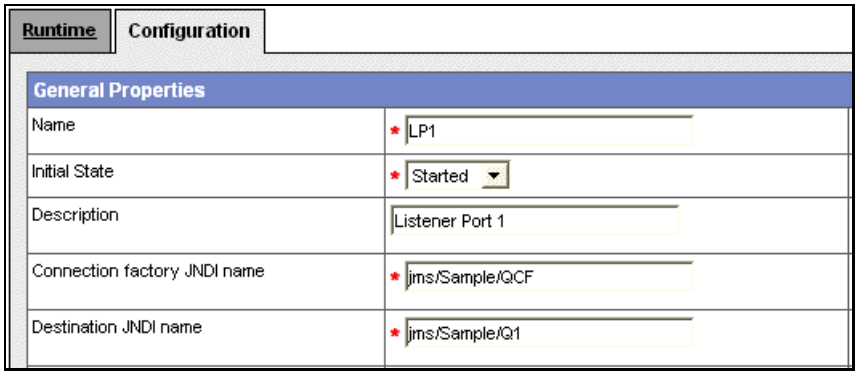


Figure 5-3 Add a new listener port

8. We now have to set the JMS server properties. Start again with a select of **Server -> Application Server** and select **your-app-server**. The following steps 9 to 14 can also be done in the server configuration of your WebSphere Studio Application Developer V5.0 test server. If you configure WebSphere Application Server, we recommend to continue with the next configuration steps.
9. In the Additional Properties table select **Server Components**. The Server Component table appears. Select **JMS Servers** and you will see the general properties for the Internal JMS Server.
10. In section General Properties for the Internal JMS Server, type the Queue name (Q0) and select **Initial State Started** from the drop-down combo box.
11. Click the **OK** button and save your changes to the master configuration.
12. To map the connection factory name and queue name to their corresponding JNDI names for the WebSphere JMS provider, select **Resources->WebSphere JMS Provider** in the Navigation tree view.
13. You should find the WebSphereJMSProvider screen with an Additional Properties table:

a. Click **WebSphere Queue Connection Factories** and click the **New** button. Provide the Configuration menu with a Name (QCF), a **JNDI Name** (jms/itso/Q0), and select **Node localhost**. Click the **OK** button and reselect **WebSphereJMSProvider**.

b. Now select the **WebSphere Queue Destinations**, click the **New** button and fill in Name (Q0), JNDI Name (jms/itso/Q0) and select **Node** localhost. You may also define additional destination queue properties like Category (if you have destination topic), Persistence, Priority, and Expiry. Finish the configuration by clicking the **OK** button.

14. To save the master configuration, select **Save** from the application task bar and conform the Save to Master Configuration dialog by clicking the **Save** button.

15. To have the changed master configuration take effect, stop and start the application server.

JMS Provider definitions

Until now we defined a listener port (LP0) and assigned the listener port to an connection factory JNDI name (jms/itso/QCF) and a destination JNDI name (jms/itso/Q0). Factory name and destination name must be defined for your V5.0 test server, if you want use the destination queue and Message-driven Beans on that server. To provide JMS definition for your test server:

1. Select the **Server Configuration** view and click the **JMS** tab. The display shows `WebSphere JMS Provider Options` (Figure 5-4).

2. Select **Server Settings** and add the destination queue names (Q0) to the JMS Server Properties.

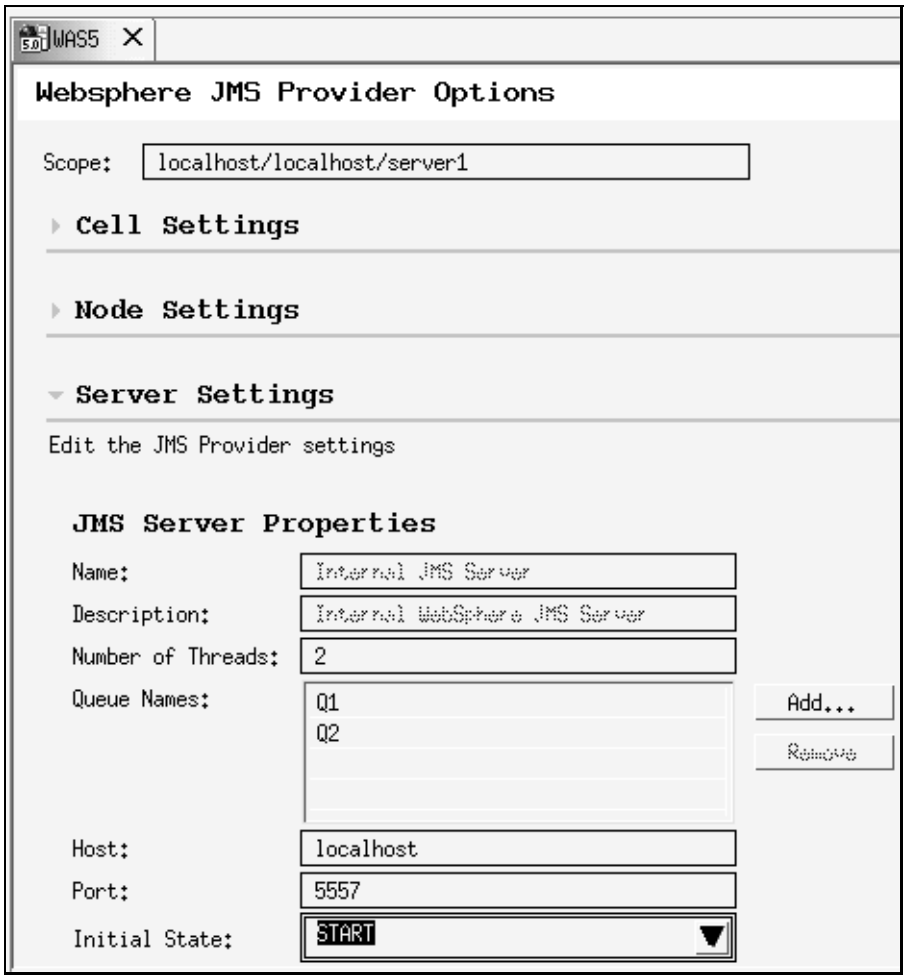


Figure 5-4 WebSphere JMS Provider options

- Click the **Add** button for the **WASQueueConnectionFactory entries** table. A dialog appears (Figure 5-5), where you can type in the following:
 - Name: the factory name (QCF)
 - JNDI name: the Connection factory JNDI name (jms/itso/QCF)
 - Server Name: Server name (server1)
 - Node: Node name (localhost)

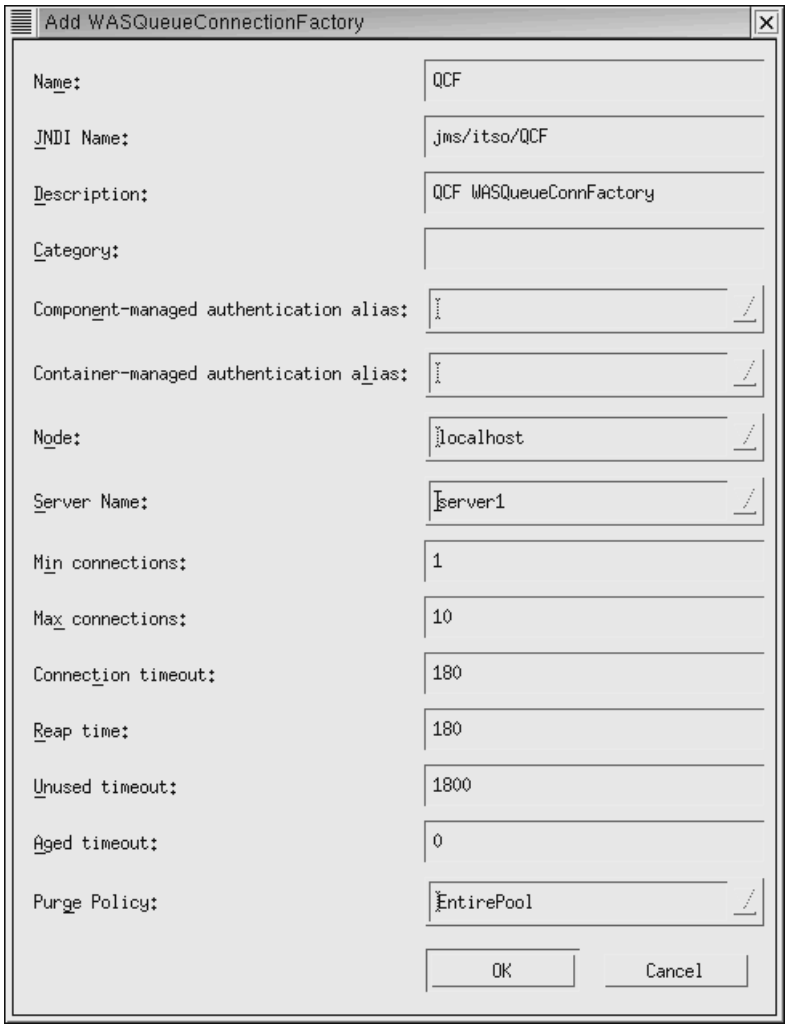


Figure 5-5 Add WASQueueConnectionFactory dialog

- Click the **Add** button for WASQueue entries table. A dialog appears (Figure 5-6) where you can type in the following:
 - Name: The queue name (Q0)
 - JNDI Name: The destination JNDI name (jms/itso/Q0)
 - Node: Node name (localhost)

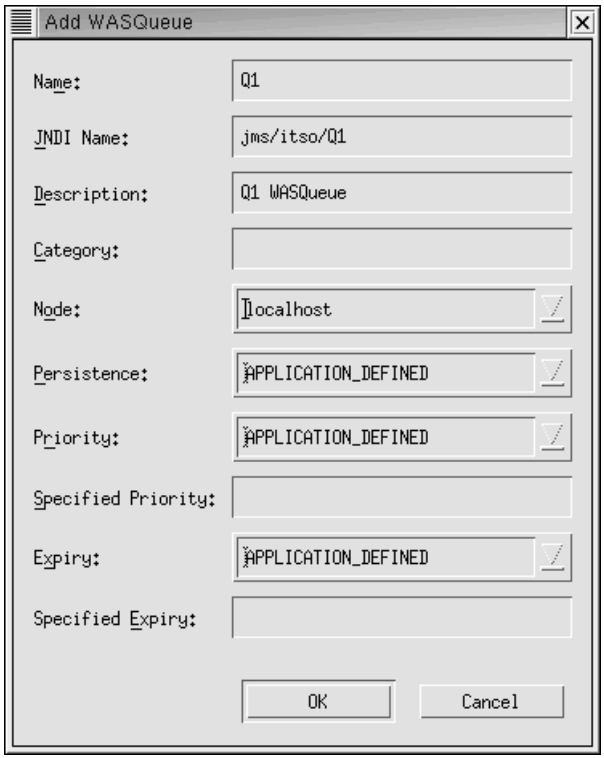


Figure 5-6 Add WASQueue dialog

- Save the server configuration and restart your test server.
- Check the console log when the test server is restarted. You should find following messages according to your definition names:
 - `MSGS0650I: MQJD JMS Provider open for business`
 - `WSVR0049I: Binding Q0 as jms/itso/Q0`
 - `WSVR0049I: Binding QCF as jms/itso/QCF`

We now have successfully defined a listener port name, which was mapped to a connection factory JNDI name and a destination JNDI name using the Admin client. We also defined factory names and queue names, which were assigned to their corresponding JNDI names using the V5.0 test server configuration.

Create a Message-driven Bean

In this chapter we describe the implementation of a Message-driven Bean, which will listen to a listener port (LP0). The bean's destination type is a queue (Q0) belonging to a connection JNDI factory (QCF).

Listener port name	Connection factory name	Connection JNDI factory name	Queue name	Queue JNDI name
LP0	QCF	jms/itso/QCF	Q0	jms/itso/Q0

Prior to start writing a message driven bean you need to have an EJB project and a package for the Message-driven Bean:

1. Create an EAR application project named ITSOBankEAR. Select **File-New Project**. In the New Project Dialog select **J2EE** in the left panel, and **Enterprise Application Project** in the right panel. Click **Next** button.
2. Specify `Create J2EE 1.3 Enterprise Application project` and click **Next**.
3. Type in the application project name **`ITSOBankEAR`**. Deselect all other options and select only the creation of the **EAR** file.
4. Within WebSphere Studio Application Developer select the **J2EE perspective** and create a new EJB project according to the EJB 2.0 specification: ITSOBankEJB.
5. Also create a package within the ejbModule folder of the EJB project called jms.itso.bean.
6. With a right-click on the **ejbModule** folder select **New -> Other...** and a dialog appears where you should select **EJB** in the left pane an enterprise bean in the right pane.
7. The next dialog step will show you the ITSOBankEJB project name. Click the **Next** button.
8. Next step after project selection shows the dialog headed with Create a 2.0 enterprise bean. Select the **Message-driven Bean** and type in the bean name: **`RcvQ0`** (Figure 5-7).

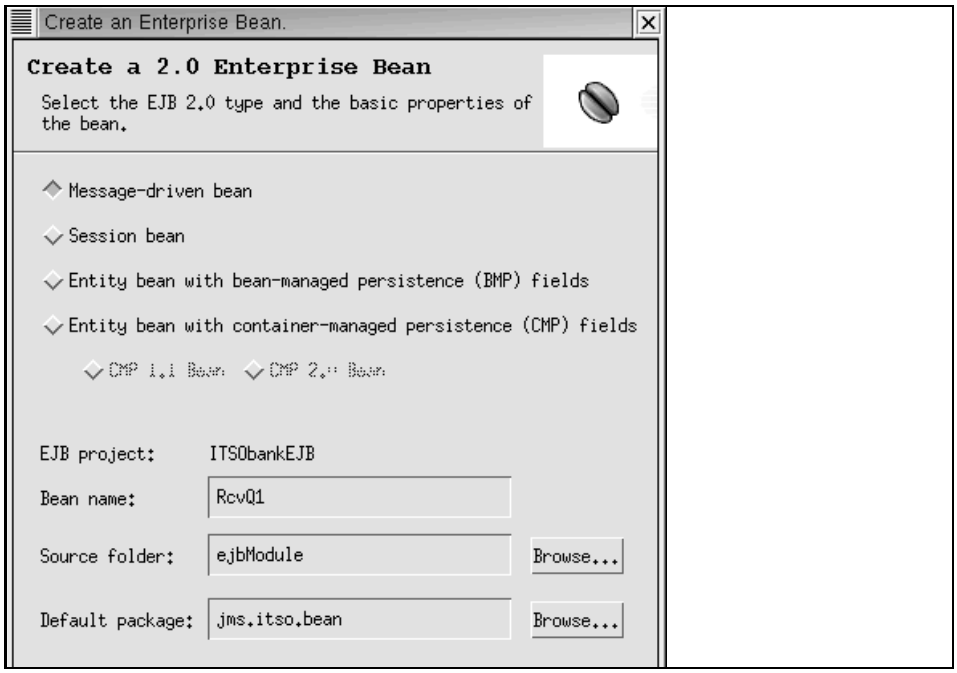


Figure 5-7 Select 2.0 EJB types

9. During the last step you have to define the property details of your Message-driven Bean (Figure 5-8):
 - Transaction type: Container or bean
 - Acknowledge mode: AutoAcknowledge or DupsOkAcknowledge
 - Destination Type: Queue or Topic. If Topic selected, you have also the choice between durable and non-durable message live time.
 - Message selector, which applies only if Destination Type Topic is selected.
 - Listener Port Name (LP0)

As a first start we selected **Bean with AutoAcknowledge** as transaction type, Queue as destination type, and the ListenerPort name: LP0. The bean name is RcvQ0 and stored in package jms.itso.bean in the ejbModule folder of the ITSOBankEJB project.

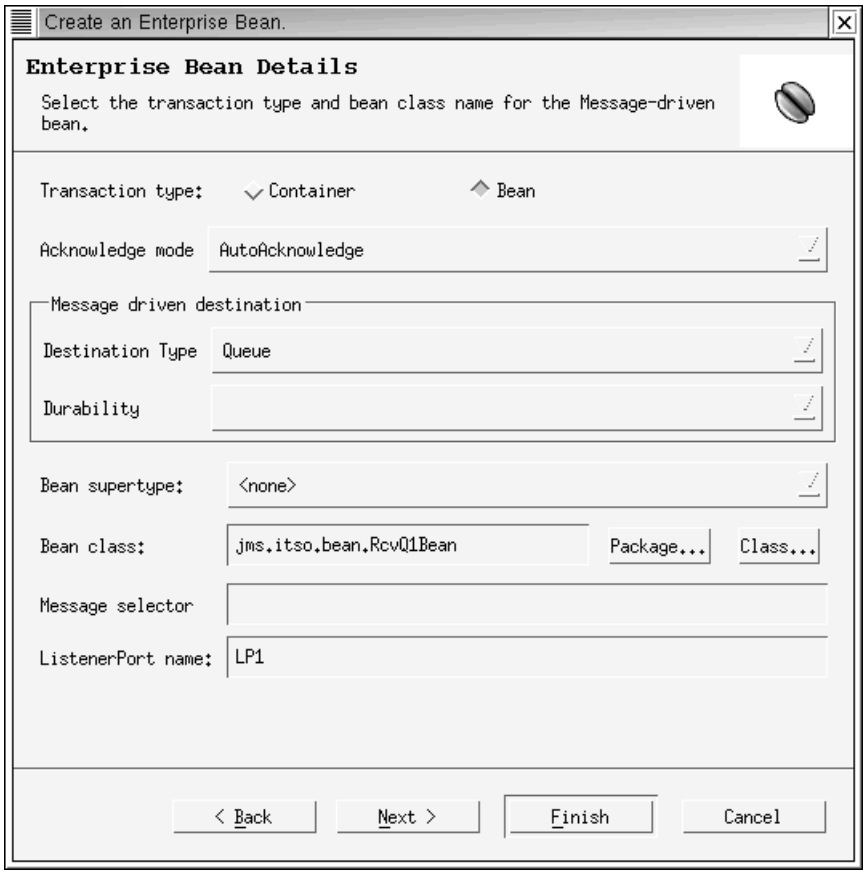


Figure 5-8 Define EJB details for a Message-driven Bean

After finishing the Create an Enterprise dialog, WebSphere Studio Application Developer generates a Message-driven Bean skeleton named RcvQ0Bean.java in package jms.itso.bean, which resides in the ejbModule folder of our ITSOBankEJB enterprise project. The RcvQ0Bean.java class contains following five methods:

- getMessageDrivenContext()
- setMessageDrivenContext(javax.ejb.MessageDrivenContext ctx)
- ejbCreate()
- onMessage(javax.jmsMessage msg)
- ejbRemove()

WebSphere Studio Application Developer also generates an entry in the EJB Deployment Descriptor file ejb-jar.xml and in the IBM extension deployment descriptor ibm-ejb-jar-bnd.xmi.

Example 5-2 EJB Deployment Descriptor entry for a Message-driven Bean

```
<enterprise-bean>
   <message-driven Id="Messagedriven_1034392674661">
      <ejb-name>RcvQ0Bean</ejb-name>
      <ejb-class>itso.test.bean.RcvQ0Bean</ejb-class>
      <transaction-type>Bean</transaction-type>
      <acknowledge-mode>Auto-acknowledge</acknowledge-mode>
      <message-driven-destination>
         <  Destination-type>javax.jms.queue</destination-type>
      </message-driven-destination>
   </message-driven>
</enterprise-bean>
```

You will find following binding information in the IBM extension deployment descriptor ibm-ejb-jar-bnd.xmi.

Example 5-3 Binding Listener Port to a Message-driven Bean

```
<ejbBindings xmi:type="ejbbnd:MessageDrivenBeanBinding"
xmi:id="MessageDrivenBeanBinding_1034392526661"
listenerInputPortName="LP0">
<enterpriseBean xmi:type="ejb:MessageDriven"
href="META-INF/ejb-jar.xml#MessageDriven_1034392482387"/>
```

We now have to receive the message from queue Q0 and to implement some business logic within the onMessage method of the RcvQ0Bean.java code. To do so, you need knowledge about the architecture of messages in JMS.

A JMS message is composed of a header, optional properties, and an optional body. The message header contains a set of fixed fields that are used to identify the message and to help in routing. Message properties are application related header fields the programmer can set only on the sending side. They are read-only for the Message-driven Bean. The message body can take five different forms:

- **TextMessage**. The message body is a Java string.
- **BytesMessage**. A bytes message supports a body with uninterpreted data. The message supports the methods of the DataInputStream and DataOutputStream interfaces from the Java I/O package.
- **StreamMessage**. A stream message adds a body that is a stream of objects.
- **ObjectMessage**. An object message provides a body that can contain any Java object that supports the serializable interface.
- **MapMessage**. A map message supports the Message interface and provides a body of name/value pairs.

The code snippet below uses the MapMessage interface. Expect receiving a formatted message.

Example 5-4 Completing the onMessage method with sample code

```
/**
* onMessage
*/
public void onMessage(javax.jms.Message inMessage) {
   javax.jms.MapMessage msg = null;

try {
   if (inMessage instanceof javax.jms.MapMessage) {
      msg = (javax.jms.MapMessage) inMessage;
      int custid = msg.getInt("Custid");
      int quantity = msg.getInt("Qnantity");
      // Implement your business logic here
      System.out.println("RcvQOBean: Custid: " + custid);
      System.out.println("RcvQOBean: Quantity: " +
         quantity);
   }
   else {
      // Act on wrong message type accordingly
      System.out.println("RcvQOBean: Wrong type.");
   }
}
catch (javax.jms.JMSException e) {
   e.printStackTrace();
   fMessageDrivenCtx.setRollbackOnly();
}
catch (Throwable te) {
   te.printStackTrace();
}
}
```

Note that the onMessage() method defined for a MD bean leaves the parsing of the inbound message up to the application programmer. The messages have to be in the right message format. If you expect many message types on the same queue, this will lead to a chain of if-statements in the Message-driven Bean.

The sample Message-driven Bean is now ready to run. The bean consumes a message from listener port LP0 which was mapped to destination queue Q0. The message type is checked and a map message with two integers (Custid and Quantity) is received and written to your WebSphere Studio Application Developer V5.0 test server console.

To test the Message-driven Bean on the WebSphere Studio Application Developer V5.0 test server, requires a message producer sending a message to the queue Q0.

Writing a JMS client

To get a better idea how JMS is used to write to a message queue, we can develop a simple JavaServer Page (JSP) whose sole purpose is to accept two integer values named Custid and Quantity as parameters, and then to send them to message queue Q0 (Figure 5-9). The command sample for sending the message within a browser then will be:

```
http://localhost:9080/ITSOBankWeb/sndQ0.jsp?C=101&Q=25
```

1. Create a Web project named ITSOBankWeb that belongs to the EAR project ITSOBankEAR.
2. Create a JSP file sndQ0.jsp in the Web Content folder of the ITSOBankWeb project.
3. Insert the Java code to request Custid and Quantity parameters.

Example 5-5 Request parameter

```
<P>Send an Check Reorder Message.</P>
<%
String param;
int custid = 0;
int quantity = 0;
try {
   param = request.getParameter("C");
   if (param != null)
      custid = Integer.parseInt(param);
   param = request.getParameter("Q");
   if (param != null)
      quantity = Integer.parseInt(param);
} catch (NumberFormatException) {}
```

4. Insert JMS code to send one message to queue Q0. Before sending a message we have to get a handle to the local JNDI environment, then look up the queue connection factory JNDI name (jms/itso/QCF) and the destination queue JNDI name (itso/jms/Q0). See this code snippet below.

Example 5-6 Send message to destination queue

```
// Send a message to Q0
javax.naming.Context ic = null;
javax.jms.QueueConnectionFactory qConnFactory = null;
javax.jms.QueueConnection qConn = null;
javax.jms.Queue queue = null;
javax.jms.QueueSession qSession = null;
```

```
javax.jms.QueueSender qSender = null;
javax.jms.MapMessage message = null;

String fName = "jms/itso/QCF";
String qName = "jms/itso/QO";
try {
   java.util.Properties env = System.getProperties();
   env.put(javax.naming.Context.INITIAL_CONTEXT_FACTORY,
   "com.ibm.websphere.naming.WsnInitialContextFactory");
   env.put(javax.naming.Context.PROVIDER_URL,
   "iiop://localhost");

   ic = new javax.naming.InitialContext(env);
   qConnFactory = (javax.jms.QueueConnectionFactory)
   ic.lookup(fName);
   queue = (javax.jms.Queue) ic.lookup(qName);
}
catch (Exception e) {
   System.err.println("JNDI lookup failed " + e);
}
try {
   qConn = qConnFactory.createQueueConnection();
   qSession = qConn.createQueueSession(false,
   javax.jms.Session.AUTO_ACKNOWLEDGE);
   qSender = qSession.createSender(queue);
   message = qSession.createMapMessage();

   // Send the Check Reorder Message
   message.setInt("Custid", custid);
   message.setInt("Quantity", quantity);
   qSender.send(message);
   qConn.close();
%>
<P>Customer=<%= custid %> <br>Quantity=<%= quantity %><br>
Check Reorder sent to ITSO Bank.<P>
<%
}
catch (Exception e) {
   System.err.println("Error on sending message " + e);
}
%>
```

5. Add the JSP sndQ0.jsp to the ITSOBankWeb Web Deployment Descriptor and map the jsp-file name to /sndQ0. The web.xml then shows following entry.

Example 5-7 JSP/Servlet Web deployment entry in web.xml

```
<servlet>
   <servlet-name>sndQ0</servlet-name>
   <display-name>sndQ0</display-name>
   <jsp-file>/sndQ0.jsp</jsp-file>
</servlet>
<servlet-mapping>
   <servlet-name>sndQ0</servlet-name>
   <url-pattern>/sndQ0</url-pattern>
   </servlet-mapping>
```

6. Add the ITSOBankEAR file to your WebSphere Studio Application Developer V5.0 test server and restart your test server. Check the console message where you should find:
 - Starting application: ITSOBankEAR
 - Preparing to start EJB jar: ITSOBankEJB
 - Starting EJB jar: ITSOBankEJB
 - Application started: ITSOBankEAR

- Start the JSP sndQ0.jsp from a browser using the following URL: `http://localhost:9080/ITSOBankWeb/sndQ0?C=101&Q=25`

7. If the WebSphere Studio Application Developer Console log shows the messages according to the code in the onMessage() method of the RcvQ0Bean.java source, you have successful written a Message-driven Bean.

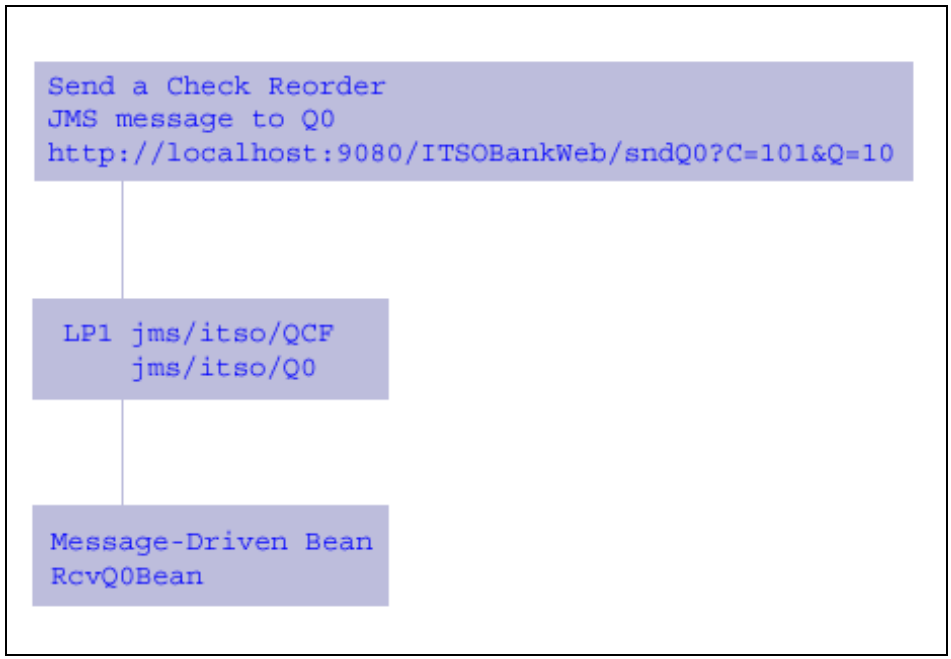


Figure 5-9 A simple asynchronous messaging scenario

Summary

Asynchronous messaging systems allow many applications, written in different languages, to exchange business information in the form of a message. Messages are transmitted from one application to another on a network using message-oriented middle ware (MOM). JMS can be used with different MOMs. For example, you can use the same JMS API to send messages with Progress' SonicMQ as with IBM's MQSeries; BEA's WebLogic JMS service; and Sun Microsystems' iPlanet Message Queue just to name a few.

EJB 2.0 Message-driven Beans act as an integration point for an EJB application. They allow to better integrate other systems into EJB applications, because the systems are loosely-coupled via simple data exchange. The Message-driven Beans are managed by the EJB container and in their onMessage() method they receive formatted messages that can be processed directly, or processing may be delegated to other Entity Beans, or passed to other message driven beans using JMS. They are an excellent addition to the EJB platform.

5.1.2 EJB 2.0 Bean Managed Persistence Entity Bean

This section describes creation of an Entity Bean with Bean-Managed Persistence (BMP). An Entity Bean implements the object view of an entity in an underlying database, or an entity implemented in an existing enterprise application; for example, by a mainframe program using flat files such as VSAM. The data access protocol for transferring the state of the entity between the bean and the database is referred to as persistence. In a BMP Entity Bean it is the developers task to write the persistence handling code into the bean class. As a database, we use the Linux version of IBM DB2 Version 7.2 in some helper classes.

The business logic in ITSO Bank sample

The ITSO Bank sample we use will realize a simple check reorder. We received two properties in our Message-driven Bean: the customer identification and the number of checks the customer wants to reorder. Customer ID and quantity will be the input for the BMP bean called by the Message-driven Bean RcvQ0Bean. The bean has to store the customer's reorder request into a check_reorder request table. The request records in the check_reorder table will enable the check print shop to produce individual checks and mail it to the customer.

Example 5-8 Check-reorder request table definition

```
CREATE TABLE ITSO.CHECK_REORDER (
   CK_RO_ID INTEGER NOT NULL GENERATED ALWAYS AS IDENTITY
      (START WITH 101, INCREMENT BY 1, NO CACHE),
   FK_CUST INTEGER NOT NULL,
   QUANTITY INTEGER NOT NULL,
   STATUS VARCHAR(15) NOT NULL,
   DATAE_ORDERED TIMESTAMP NOT NULL,
   PRIMARY KEY (CK_RO_ID));
ALTER TABLE ITSO.CHECK_REORDER
   ADD CONSTRAINT "CheckReordering"  FOREIGN KEY (FK_CUST)
   REFERENCES ITSO.CUSTOMER ON DELETE RESTRICT;
GRANT ALL ON ITSO.CHECK_REORDER TO PUBLIC;
```

The function for the BMP bean is to insert a customer check reorder into the check_reorder request table of the WSAD5 database. A new request ID will be generated by incrementing the maximum check_reorder primary key CK_RO_ID. The CheckReordering constraint will guarantee that inserted records have an existing customer ID. If we get an invalid customer ID delivered from the Message-driven Bean, then the check_reorder insert into the request table will fail.

The BMP bean will manage following value objects:

- ckroID - An integer that represents the request ID. We define valid request IDs that are not zero.
- custID - An integer representing the customer ID. Valid customer IDs are not zero.
- quantity - An integer representing the amount of checks reordered.
- status - A string describing the status of the check reorder. For example: received, printed, or mailed.
- date_ordered - A timestamp marking the last status.
- customer - A string with title, surname, and last name of the customer. The bean will retrieve the customer string from the customer table of the ITSO Bank sample.

First we will write the business objects and the getter and setter methods for our objects into a Java class named CheckVO, representing the check reorder business, and managing value exchange with the bean.

Example 5-9 CheckVO.java: Value objects for Check-Reorder

```
package itso.bank.ejb;
import java.sql.Timestamp;
public class CheckVO implements java.io.Serializable {
   private int ckroID;
   private int custID;
   private int quantity;
   private String status;
   private String acctype;
   private Timestamp date_ordered;
   private String customer;
   /**
    * Gets the ckroID
    * @return Returns an Integer
    **/
   public int getCkroID() {
      return ckroID;
   }
   /**
   * Sets the ckroID
   * @param int ckroID The ckroID to set
   */
   public void setCkroID(int ckroID) {
      this.ckroID = ckroID;
   }
   /**
    * Gets the custID
```

```
 * @return Returns an Integer
 */
public int getCustID() {
   return custID;
}
/**
 * Sets the custID
 * @param int custID The custID to set
*/
public void setCustID(int custID) {
   this.custID = custID;
}
/**
 * Gets the quantity
 * @return Returns an Integer
 */
public int getQuantity() {
   return quantity;
}
/**
 * Sets the quantity
 * @param int quantity The quantity to set
 */
public void setQuantity(int quantity) {
   this.quantity = quantity;
}
/**
 * Gets the status
 * @return Returns a String
 */
public String getStatus() {
   return status;
}
/**
 * Sets the status
 * @param String status The status to set
*/
public void setStatus(String status) {
   this.status = status;
}
/**
 * Gets the date_ordered
 * @return Returns a java.sql.Timestamp
 */
public Timestamp getDate_ordered() {
   return date_ordered;
}
/**
 * Sets the date_ordered
```

```
     * @param Timestamp date_ordered
     */
    public void setDate_ordered(Timestamp date_ordered) {
        this.date_ordered = date_ordered;
    }
    /**
     * Gets the customer
     * @return Returns a String
     */
    public String getCustomer() {
        return customer;
    }
    /**
     * Sets the customer
     * @param String customer The customer to set
     */
    public void setCustomer(String customer) {
        this.customer = customer;
    }
}
```

Creating a BMP Entity Bean

To create the bean-managed persistence entity for check reorder follow the steps below:

1. Select **File->New--Enterprise Bean** in the WebSphere Studio Application Developer V5.0 task bar. The Enterprise Bean Creation dialog box will appear. Select the **ITSOBankEJB** enterprise bean and click **Next**.
2. This step asks for the type of the 2.0 enterprise bean we want to create. Select **Entity bean with bean-managed persistence (BMP)** and type in the following:
 - Bean name: Check
 - Source Folder: ejbModule
 - Default package: jms.itso.bean
 - Then click **Next**.
3. Select **Local client view** and **Remote client view**. Then click **Next** and **Finish**.

You will find four EJB interfaces and two EJB classes in the package jms.itso.bean:

- **Check.java**: The remote interface that extends the javax.ejb.EJBObject. Here is the place where we can define the business methods that can be accessed from the outside world.

- **EJB 2.0 CheckLocal**.java: The local interface extending the EJBLocalObject. Here we can define the business methods the bean presents to other beans in the same address space and same EAR file. It allows the bean to interact without the overhead of a distributed object protocol.
- **CheckHome**: The home interface that extends EJBHome.
- **EJB 2.0 CheckLocalHome**: The local home interface extending EJBLocalHome.
- **CheckBean**: The bean class that implements the Entity Bean.
- **CheckKey**: The bean class for the primary key of the bean, which is a pointer into the database.

We now can implement data exchange with the Check bean using the CheckVO value objects:

1. First we will insert the *set* and *get* methods of our business properties defined in CheckVO.java into the Entity Bean CheckBean. Insert following code snippet into CheckBean.java.

Example 5-10 Defining 'get' and 'set' methods for value object CheckVO

```
   public jms.itso.bean.CheckVO checkVO = null;
   /**
    * getCheckVO
    */
   public CheckVO getCheckVO() {
      return checkVO;
   }
   /**
    * setCheckVO
    */
   public void setCheckVO(CheckVO checkVO) {
      this.checkVO = checkVO;
}
```

2. To interact with a client we have to pass the CheckVO value objects to the ejbCreate() and ejbPostCreate() method as an argument. We copy the incoming value objects into the bean using **`setCheckVO(checkVO);`**

 In addition, we print out informational messages in the ejbCreate() method of the CheckBean to the WebSphere Studio Application Developer V5.0 test server console log as a simple check if whether processing and interaction between the calling EJB and the Check bean works properly. Note that the ejbCreate() method has to return the primary key, which is the check reorder ID int ckroID in our sample.

Example 5-11 Defining arguments for ejbCreate and ejbPostCreate

```
   /**
    * ejbCreate
    */
   public jms.itso.ejb.CheckKey ejbCreate(CheckVO checkVO)
      throws javax.ejb.CreateException {
      setCheckVO(checkVO);
      // Insert check reorder into table
      // ...
      // Write to the console log
      System.out.println("CheckBean:ejbCreate:CkroID=" +
         checkVO.getCkroID());
      System.out.println("CheckBean:ejbCreate:CustID=" +
         checkVO.getCustID());
      System.out.println("CheckBean:ejbCreate:Quantity="
         + checkVO.getQuantity());
      System.out.println("CheckBean:ejbCreate:Status=" +
         checkVO.getStatus());
      System.out.println("CheckBean:ejbCreate:Time=" +
         checkVO.getDate_ordered());
      System.out.println("CheckBean:ejbCreate:Customer=" +
         checkVO.getCustomer());
      return new CheckKey(ckroID);
   }
   /**
    * ejbPostCreate
    */
   public void ejbPostCreate(CheckVO checkVO)
      throws javax.ejb.CreateException {
   }
...
```

3. The CheckVO value class needs to be declared as an argument in the create interface of CheckHome and CheckLocalHome. In CheckHome.java add an argument the create definition as follows:
 - public jms.itso.bean. Check **create(CheckVO checkVO)** throws javax.ejb.CreateException, java.rmi.RemoteException;
 - In CheckLocalHome.java you have to do the same. Note that the local home interface does not throw the java.rmi.RemoteException, because we can interact with this interface only locally without RMI.
4. At last we have to promote the setCheckVO and getCheckVO methods in the remote and local interfaces Check.java and CheckLocal.java. Insert the definitions of Example 5-12 into both Java files. In the local interface CheckLocal.java, we can remove throwing the RemoteException. The only differences between the remote and local interface are that the local interface

does not throw a remote exception and it extends the parent interface EJBLocalObject.

Example 5-12 Method definitions in the bean's remote interface

```
package jms.itso.bean;
/**
 * Remote interface for Enterprise Bean: Check
 */
public interface Check extends javax.ejb.EJBObject {
   /**
    * setCheckVO
    */
   public void setCheckVO(CheckVO checkVO) throws java.rmi.RemoteException;
   /**
    * getCheckVO
    */
   public CheckVO getCheckVO() throws java.rmi.RemoteException;
}
```

The primary key of an Entity Bean is a pointer into the database that helps to locate a unique data record very fast. Only Entity Beans do have a primary key, for session bean and Message-driven Beans they do not make sense. The primary key is used in the findByPrimaryKey() method of the home interface CheckHome. We will implement the check reorder request number presented by the integer ckroid as a primary key together with the getCkroID() and setCkroID() methods in the CheckKey.java file.

Example 5-13 Defining a primary key

```
public class CheckKey implements java.io.Serializable {
//...
   int ckroid;
   /**
    * get CK_RO_ID
    */
   public int getCkroID() {
      return ckroid;
   }
   /**
    * set a new CK_RO_ID, same as CheckKey(int )
    */
   public void setCkroID(int ckroid) {
      this.ckroid = ckroid;
   }
   /**
    * Creates an CK_RO_ID key for Entity Bean: Check
    */
   public CheckKey(int ckroid) {
```

```
        this.ckroid = ckroid;
    }
...
}
```

We now have defined the value objects of the check reorder business object in **CheckVO.java** and we created the BMP Entity Bean Check. In **CheckBean.java** we declared the value class CheckVO and we defined the setCheckVO and getCheckVO methods. We implemented the CheckVO value class as an argument for the ejbCreate() and ejbPostCreate() method in **CheckBean.java** and some test code was applied in ejbCreate(). The ejbCreate() method returns the primary key we created. The primary key will be later used in the ejbLoad() method using getPrimaryKey().

We also defined CheckVO as an argument in the corresponding create statement of the remote home and local home interfaces **CheckHome.java** and **CheckLocalHome.java**, so that they have the same signature. The getCheckVO and setCheckVO methods were defined in the remote and local interfaces **Check.java** and **CheckLocal.java** in order to use these methods from the outside world and to set actual values or retrieve resulting values from the bean. The primary key check reorder request id was implemented as **int ckroid** in **CheckKey.java** together with a setter and getter method for interchange purposes with the CheckKey class.

We are now ready to interact with bean's ejbCreate() method using the CheckVO value objects. Since we intend to use the findByPrimaryKey() and ejbLoad() methods, we have to implement the primary key into this methods for fast access of data rows and for loading the data into our value objects. Within the ejbLoad() method, the primary key is addressed with the help of the bean's entity context.

Example 5-14 PrimaryKey in ejbFindByPrimaryKey() and ejbLoad() methods

```
    /**
     * ejbFindByPrimaryKey
     */
    public jms.itso.bean.CheckKey ejbFindByPrimaryKey(
        jms.itso.bean.CheckKey primaryKey)
        throws javax.ejb.FinderException {
        System.out.println(
        "CheckKey:ejbFindByPrimaryKey: START");
        int ckroID = primaryKey.getCkroID();
        // Select data with primary key ckroID
        ...
        if (ckroID != 0) {
            return primaryKey;
        }
        else {
```

```
            throw new javax.ejb.FinderException();
        }
    }
    /**
     * ejbLoad
     */
    public void ejbLoad() {
        System.out.println("CheckBean:ejbLoad: START");
        CheckKey primaryKey =
        (CheckKey) myEntityCtx.getPrimaryKey();
        // Load data with primary key
        ...
    }
```

If the ITSO Bank sample would require to delete check reorders, we would implement the statement CheckKey primaryKey = (CheckKey) myEntityCtx.getPrimaryKey(); also into the ejbRemove() method.

We now have a BMP bean only with value objects that exists in the Java runtime storage. We have to persist the Entity Bean, which means we have to write the check reorders into the check_reorder table using JDBC and SQL. Until now there is no connection to our underlying database WSAD5, where we can make our check reorders persistent. Since we are writing a bean-managed persistence Entity Bean, we must write code to translate our object values into the check_reorder table of the WSAD5 database using JDBC and SQL with DB2. The next chapter describes how we persist the check bean.

Implement database access objects

In this chapter we demonstrate the implementation of access code to an underlying database for an Entity Bean. We assume that the WSAD5 database already exists and that the required tables (check_reorder and customer) are defined and populated with some data. We will define SQL statements to select, insert, update, and delete rows from the check_reorder table to achieve persistence for the Check bean. To make our data access classes more independent, we put all application related data into an interface named Constants.java. In this interface we stored the following items:

- The data source JNDI name: jdbc/WSAD5
- The naming context host name: localhost
- The naming context host port: 2809
- The user ID and password to access the database
- The naming context factory class of the WebSphere Studio Application Developer V5.0 test server:
 `com.ibm.websphere.naming.WsnInitialContextFactory`

- SQL prepared statements to perform select, insert, update and delete rows as required to make the bean persistent:
 - INSERT INTO ITSO.CHECK_REORDER (FK_CUST, QUANTITY, STATUS, DATE_ORDERED) VALUES(?, ?, ?, CURRENT TIMESTAMP) for insert a new request reorder row.
 - SELECT IDENTITY_VAL_LOCAL() FROM SYSIBM.SYSDUMMY1 is required to get the last recent check reorder request id after a new row was inserted into the check_reorder table.
 - SELECT TITLE, FIRSTNAME, LASTNAME FROM ITSO.CUSTOMER WHERE CUSTOMERID = ? for load of the customer value object.
 - SELECT CK_RO_ID FROM ITSO.CHECK_REORDER WHERE CK_RO_ID = ? to find a check_reorder row for a given primary key within the bean.
 - SELECT FK_CUST, QUANTITY, STATUS, DATE_ORDERED, TITLE, FIRSTNAME, LASTNAME FROM ITSO.CHECK_REORDER, ITSO.CUSTOMER WHERE WHERE CK_RO_ID = ? AND FK_CUST = CUSTOMERID for load of the checkVO value objects with one select statement.
 - UPDATE ITSO.CHECK_REORDER SET QUANTITY = ?, STATUS = ?, DATE_ORDERED = CURRENT TIMESTAMP WHERE CK_RO_ID = ? to change quantity and status of a request reorder row.
 - DELETE FROM ITSO.CHECK_REORDER WHERE CK_RO_ID = ? to delete of a request reorder row.
 - The JNDI names of the remote home and local home interface.
 - The JNDI names of the queue connection factory and the queues we will use to run the ITSO Bank sample BMP Entity Bean called from a Message-driven Bean.

The methods we use for managing data access to the WSAD5 database are defined in a data access object interface named **CheckDAO.java** and the implementation of the data access methods for DB2 is coded in **DB2CheckDAO.java**. The check reorder ID is the input value for following methods:

- **public int findCheck(int ckroID)**, which returns a valid check reorder id if it exists or zero if no valid check reorder id is found in the check_reorder table
- **public CheckVO getCheck(int ckroID)** requests the check_reorder table and customer table and returns all value objects in CheckVO
- **public boolean deleteCheck(int ckroID)** deletes a row in check_reorder table and returns true if the record was successful deleted, otherwise false is returned.

The CheckVO value objects are passed as an argument for the following data access methods:

- public int insertCheck(CheckVO checkVO). This method selects the maximal check reorder id from the check_reorder table and increments this value by one. Then a new row is inserted into the table using customerid, quantity and status from CheckVO. A new current timestamp is used for the date_ordered field. The new check reorder id is returned to the calling routine.
- public boolean updateStatus(CheckVO checkVO) updates the quantity, the status and the date_ordered timestamp. The method returns true, if the update was successful, otherwise it returns false.

The connection to the WSAD5 database we are using in the ITSO Bank sample is built in the **ConnectionPoolHelper.java**. This utility class looks up the data source name and returns one instance of the connection to the WSAD5 database. It uses the singleton design pattern, so that we have never more than one connection to the database. Connectivity to the data source is created in a synchronized method during the first demand for connection to WSAD5.

We have implemented the abstract class **DAOFactory**, where we can define more data access objects like the CheckDAO or other data access factories than DB2. The DB2 factory is coded in the class **DB2DAOFactory**, where we can get a connection to the WSAD5 database and the check_reorder data access object **DB2CheckDAO**. A new instance of the CheckDAO for DB2 is called with CheckDAO checkDAO = DAOFactory.getDAOFactory(DAOFactory.DB2_DAO).getCheckDAO();. If the data source is not found, an exception is thrown coded in **DataSourceNotFound.java**. The table below gives an overview of all data access modules the ITSO Bank sample uses for DB2 access to WSAD5 database.

Table 5-1 Data access modules for DB2

Data access module	Functional description
Constants.java	Installation dependent variable like JNDI names, queue factory names and queue names, data source names, user ID and password for DB2, etc.
ConnectionPoolHelper.java	Delivers one connection to the data source using JNDI naming service.
DataSourceNotFoundException.java	Extends exception and indicates that the database could not be found
CheckDAO.java	The data access object interface
DB2CheckDAO.java	The DB2 data access object contains the java coding of the data manipulation language
DB2DAOFactory	The class extends the DAOFactory for DB2. Returns a connection and new DB2 data access object
DAOFactory	Abstract class to get a vendor specific DAOFactories and data access objects

Each method in DB2CheckDAO closes the connection at the end of the method, allowing the EJB container to pool JDBC connections. The implementation of the data access objects makes use of connection pooling, which is a built-in function in the JDBC 2.0 specification and happens automatically. The data access objects performs the data manipulation with prepared statements. connection pooling and prepared statements are important performance improver. Prepared statements are cached by DB2 when executed on a connection. If a prepared statement is executed, DB2 looks up the cache and if the statement has been executed previously, it reuses the previous prepared version, which improves processing time.

Persist the bean

In this chapter we will finish the coding of the BMP Entity Bean. All what remains to do is to *persist the bean*. Because the bulk of JDBC code required for the methods to perform CRUD (create, read, update, delete) operations is already done in the DB2CheckDAO data access module, we only have to insert a few lines of code into the CheckBean.java source.

First we have to define the data access object CheckDAO and to create an instance of our data access object. This is done in the setEntityContext() method

and the data access object is released in the unsetEntityContext within the CheckBean.

Example 5-15 Implement data access object CheckDAO into the CheckBean

```
package jms.itso.bean;
/**
 * Bean implementation class for Enterprise Bean: Check
 */
public class CheckBean implements javax.ejb.EntityBean {
   private javax.ejb.EntityContext myEntityCtx;
   public jms.itso.bean.CheckDAO checkDAO = null;
   public jms.itso.bean.CheckVO checkVO = null;
   public javax.ejb.EntityContext getEntityContext() {
      return myEntityCtx;
   }
   /**
    * setEntityContext
    */
   public void setEntityContext(javax.ejb.EntityContext ctx) {
      myEntityCtx = ctx;
      System.out.println(
         "CheckBean:setEntityContext(): START");
      // Get data access objects
      checkDAO =
      DAOFactory.getDAOFactory(1).getCheckDAO();
   }
   /**
    * unsetEntityContext
    */
   public void unsetEntityContext() {
      myEntityCtx = null;
      // Release data access objects
      checkDAO = null;
   }
...

}
```

In **ejbCreate(Check checkVO)** we get custid and quantity from the argument checkVO. The incoming values are copied into the bean. Then we have to insert a record into the check_reorder table and to query the customer table for title, first name, and last name of the incoming custid. The new requestid and the customer info are copied to the beans value objects. The bean's primary key is returned.

Example 5-16 Persist new records in ejbCreate()

```
/**
 * ejbCreate
 */
public jms.itso.bean.CheckKey ejbCreate(CheckVO checkVO)
   throws javax.ejb.CreateException {
   setCheckVO(checkVO);
   int ckroID = checkDAO.insertCheck(checkVO);
   System.out.println(
   "CheckBean:ejbCreate(): New Requestid: " + ckroID);
   checkVO.setCkroID(ckroID);
   String customer =
      checkDAO.findCustomer(checkVO.getCustID());
   checkVO.setCustomer(customer);
   // Write to the console log
   System.out.println("CheckBean:ejbCreate:CkroID=" +
      checkVO.getCkroID());
   System.out.println("CheckBean:ejbCreate:CustID=" +
      checkVO.getCustID());
   System.out.println("CheckBean:ejbCreate:Quantity=" +
      checkVO.getQuantity());
   System.out.println("CheckBean:ejbCreate:Status=" +
      checkVO.getStatus());
   System.out.println("CheckBean:ejbCreate:Time=" +
      checkVO.getDate_ordered());
   System.out.println("CheckBean:ejbCreate:Customer=" +
      checkVO.getCustomer());
   return new CheckKey(ckroID);
   }
```

It would be also possible to complete the customer information in the ejbPostCreate() method, called by the container after the ejbCreate() method. We then would have to retrieve the customer field with checkDAO.findCustomer(checkVO.getCustID)) method and to complete the bean's value object within ejbPostCreate().

The **ejbFindByPrimaryKey()** method in the CheckBean class has to find a primaryKey. The primaryKey is given by the calling routine with the CheckKey(int) method. The method returns the primaryKey. If no record is found a FInderException is thrown.

Example 5-17 ejbFindByPrimaryKey() method in CheckBean

```
/**
 * ejbFindByPrimaryKey
 */
public jms.itso.bean.CheckKey ejbFindByPrimaryKey(
```

```
    jms.itso.bean.CheckKey primaryKey)
    throws javax.ejb.FinderException {
    System.out.println(
       "CheckKey:ejbFindByPrimaryKey: START");
    int ckroID =
       checkDAO.findCheck(primaryKey.getCkroID());
    if (ckroID != 0) {
       return primaryKey;
    }
    else {
       throw new javax.ejb.FinderException();
    }
}
```

The **ejbLoad()** method in the CheckBean class is responsible to load a row from the database table. By using getPrimaryKey() in ejbLoad() it is known which row to load from the table. There is no return code from this method.

Example 5-18 ejbLoad() method in CheckBean

```
/**
 * ejbLoad
 */
public void ejbLoad() {
   System.out.println("CheckBean:ejbLoad: START");
   CheckKey primaryKey =
      (CheckKey) myEntityCtx.getPrimaryKey();
   checkVO = checkDAO.getCheck(primaryKey.getCkroID());
   // Write to the console log
   System.out.println("CheckBean:ejbLoad:CkroID=" +
      checkVO.getCkroID());
   System.out.println("CheckBean:ejbLoad:CustID=" +
      checkVO.getCustID());
   System.out.println("CheckBean:ejbLoad:Quantity=" +
      checkVO.getQuantity());
   System.out.println("CheckBean:ejbLoad:Status=" +
      checkVO.getStatus());
   System.out.println("CheckBean:ejbLoad:Time=" +
      checkVO.getDate_ordered());
   System.out.println("CheckBean:ejbLoad:Customer=" +
      checkVO.getCustomer());
}
```

ejbStore() saves the bean instance's object values into the underlying database. It is the complement of ejbLoad(). The container worries about the proper time to call ejbLoad() or ejbSave(). The bean should be prepared to accept an ejbLoad() and ejbStore() at almost any time. This is one advantage of an EJB, you do not have to worry about synchronizing the bean's objects with the underlying

database. In our ITSO Bank bean sample, there was no need to implement ejbStore(). Because we only inserted rows and loaded rows into and from the database table check-reorder, an update of the database rows was not necessary. If we have to implement the ejbStore() method, then we should introduce *dirty markers* to track change of the object value and indicating that an update of the row is required.

To finish building the BMP Entity Bean Check, we have to deploy and generate RMIC code: Select the **J2EE Navigator** perspective and click the **ITSOBankEJB** project. Right-click **Generate -> Deploy and RMIC Code...** A dialog appears where you have to click the checkbox for the **Check** Entity Bean. Click the **Finish** button, and WebSphere Studio Application Developer V5.0 starts generating RMIC code for the BMP Entity Bean Check.

Deployment descriptor

Now let us take a look at the deployment descriptor, shown below.

Example 5-19 The CheckBean's deployment descriptor ejb-jar.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE ejb-jar PUBLIC "-//Sun Microsystems, Inc.//DTD Enterprise JavaBeans
2.0//EN" "http://java.sun.com/dtd/ejb-jar_2_0.dtd">
<ejb-jar id="ejb-jar_ID">
   <display-name>ITSOBankEJB</display-name>
   <enterprise-beans>
      <entity id="Check">
         <ejb-name>Check</ejb-name>
         <home>jms.itso.bean.CheckHome</home>
         <remote>jms.itso.bean.Check</remote>
         <local-home>jms.itso.bean.CheckLocalHome
         </local-home>
         <local>jms.itso.bean.CheckLocal</local>
         <ejb-class>jms.itso.bean.Checkean</ejb-class>
         <persistence-type>Bean</persistence-type>
         <prim-key-class>jms.itso.bean.CheckKey
            </prim-key-class>
         <reentrant>False</reentrant>
      </entity>
...
   </enterprise-beans>
</ejb-jar>
```

The persistence-type element indicates whether we are Bean-Managed Persistent or Container-Managed Persistent. For Bean-Managed Persistent, set it to Bean.

The prim-key-class element specifies our primary key class jms.itso.beanCheckKey.

The reentrant element dictates whether our bean can call itself through another bean. A bean is re-entrant if it calls another bean, which the calls back our bean.

Write client code calling the bean

In this chapter we will have a look from the client's point of view, who wants to use the BMP Entity Bean. The client is our Message-driven Bean RcvQ0, where we received the custid and the quantity of check reorders the customer wants to receive from the ITSO Bank. The Message-driven Bean will delegate the database work to our bean. To call a bean typically looks like this:

- Look up the home object of the bean.
- Use the home object to create an EJB object.
- Call the business method.
- Remove the EJB object.

Whenever we have to lookup for a resource in a network, we use the Java Naming and Directory Interface (JNDI). Before we can use JNDI, the initial context is required as a first starting point for JNDI services. The JNDI service delivers a reference to the home object of the bean, if the bean name is found. Because a bean is a RMI-IIOP object we need to cast it as a remote object. The static method PortableRemoteObject with the narrow() operation casts the JNDI name to a RMI-IIOP interface object. If the calling EJB and the Entity Bean resides in the same Java Virtual Machine and the same EAR-file, there is no need for using a RMI-IIOP interface. Simply use a lookup for the home name of the Entity Bean.

Example 5-20 Calling the BMP Entity Bean Check from the MDB RcvQ0Bean

```
package jms.itso.bean;

/**
 * Bean implementation class for Enterprise Bean: RcvQO
 */
public class RcvQOBean
   implements javax.ejb.MessageDrivenBean,
      javax.jms.MessageListener {
   private javax.ejb.MessageDrivenContext fMessageDrivenCtx;
   private javax.naming.Context jndiContext;
   private jms.itso.bean.CheckHome home;
   private jms.itso.bean.CheckVO checkVO;
   private jms.itso.bean.CheckVO newcheckVO;
   private jms.itso.bean.Check bean;
   /**
    * RcvQOBean
```

```
 */
public RcvQOBean() {
   System.out.println("RcvQOBean: RcvQOBean()");
}
/**
 * getMessageDrivenContext
 */
public javax.ejb.MessageDrivenContext
   getMessageDrivenContext() {
   System.out.println(
      "RcvQOBean: getMessageDrivenContext()");
   return fMessageDrivenCtx;
}
/**
 * setMessageDrivenContext
 */
public void setMessageDrivenContext(
   javax.ejb.MessageDrivenContext ctx) {
   System.out.println(
      "RcvQOBean: setMessageDrivenContext()");
   fMessageDrivenCtx = ctx;
   try {
      jndiContext =
      new javax.naming.InitialContext();
   }
   catch (javax.naming.NamingException e) {
      System.out.println(
      "RcvQOBean: Could not create JNDI context: " + e.toString());
   }
   try {
      java.util.Properties env =
         System.getProperties();
      env.put(
      javax.naming.Context.INITIAL_CONTEXT_FACTORY,
      Constants.NAMING_CONTEXT_FACTORY_CLASS);
      env.put(
      javax.naming.Context.PROVIDER_URL,
      "iiop://" + Constants.NAMING_CONTEXT_HOST_NAME
      + ":" + Constants.NAMING_CONTEXT_PORT_NUMBER
      + "/");
      jndiContext =
         new javax.naming.InitialContext(env);
      home = (jms.itso.bean.CheckHome)
         jndiContext.lookup(
         Constants.CHECK_HOME_NAME);
   }
   catch (Exception ex) {
      System.out.println(
         "RcvQOBean: Runtime Error:");
```

```
            ex.printStackTrace();
        }
    }
    /**
     * ejbCreate
     */
    public void ejbCreate() {
        System.out.println("RcvQOBean: ejbCreate()");
    }
    /**
     * onMessage
     */
    public void onMessage(javax.jms.Message msg) {
        // Receive queue messages and pass to next queue
        javax.jms.MapMessage inMessage = null;
        try {
            if (msg instanceof javax.jms.MapMessage) {
                inMessage = (javax.jms.MapMessage) msg;
                // Get Customerid and Quantity
                int custid =
                    inMessage.getInt("Custid");
                int quantity =
                    inMessage.getInt("Quantity");
                // Insert a new check reorder
                System.out.println(
                    "RcvQOBean: ---> CheckBean");
                checkVO = new jms.itso.bean.CheckVO();
                checkVO.setCustID(custid);
                checkVO.setQuantity(quantity);
                checkVO.setStatus("received");

                bean = home.create(checkVO);

                newcheckVO =
                    new jms.itso.bean.CheckVO();
                newcheckVO = bean.getCheckVO();
                System.out.println(
                "RcvQOBean:CheckBean: CkroID = "
                    + newcheckVO.getCkroID());
                System.out.println(
                    "RcvQOBean:CheckBean: CustID = "
                    + newcheckVO.getCustID());
                System.out.println(
                    "RcvQOBean:CheckBean:Quantity="
                    + newcheckVO.getQuantity());
                System.out.println("
                    RcvQOBean:CheckBean: Status = "
                    + newcheckVO.getStatus());
                System.out.println(
```

```
                "RcvQOBean:CheckBean:Customer="
                + newcheckVO.getCustomer());
             System.out.println(
                "RcvQOBean:CheckBean: Time = "
                + newcheckVO.getDate_ordered());
          }
          else {
             System.out.println("RcvQOBean: “ +
                “Message of wrong type: " +
                inMessage.getClass().getName());
          }
       }
       catch (javax.jms.JMSException e) {
          e.printStackTrace();
          fMessageDrivenCtx.setRollbackOnly();
       }
       catch (Throwable te) {
          te.printStackTrace();
       }
    }
    /**
     * ejbRemove
     */
    public void ejbRemove() {
       System.out.println("RcvQOBean: ejbremove()");
    }
}
```

Summary

BMP beans gives the application ultimate control over JDBC, and thus you have unlimited flexibility for how you map your objects to the database. Most people would love to go with CMP beans, but are afraid of the risk if it is not flexible enough. With WebSphere Studio Application Developer V5.0, it is easy to develop BMP beans, if we have data access routines available. Sending output to the console log or other logging or tracing systems allows you to understand the methods that are used by the container, and the methods that are causing bottlenecks, such as repeated loads and stores. If you start your server in debug mode, you may step through the BMP bean and see how it works.

In this chapter we walked through following steps to write a BMP Entity Bean:

- We defined of business logic the ITSO Bank bean sample for check reorder.
- We coded our value objects and their getters and setters in CheckVO.
- With WebSphere Studio Application Developer V 5.0, we generated a BMP Entity Bean Check and implemented the CheckVO value object into the bean

methods as an argument and with a set and get method. This enables us to interact with the bean.

- We defined the primary key of the bean and implemented the primary key in the CheckKey class.
- We wrote data access methods and defined SQL statements to persist our value object into an underlying database.
- We applied the data access methods in bean's methods ejbCreate(), ejbFindByPrimaryKey() and ejbLoad().
- We generated RMIC code for deployment.
- We implemented client code into the Message-driven Bean RcvQ0Bean to make use of the BMP Entity Bean managing check reorders.

5.2 ITSO Bank bean sample

The ITSO Bank bean example builds a check reorder application (Figure 5-10). The application is stored in the ITSOBankEAR.ear file and uses a HTML page, JSPs, one servlet to build a jpeg, Message-driven Beans for information exchange, and one BMP Entity Bean for the database related work of the application.

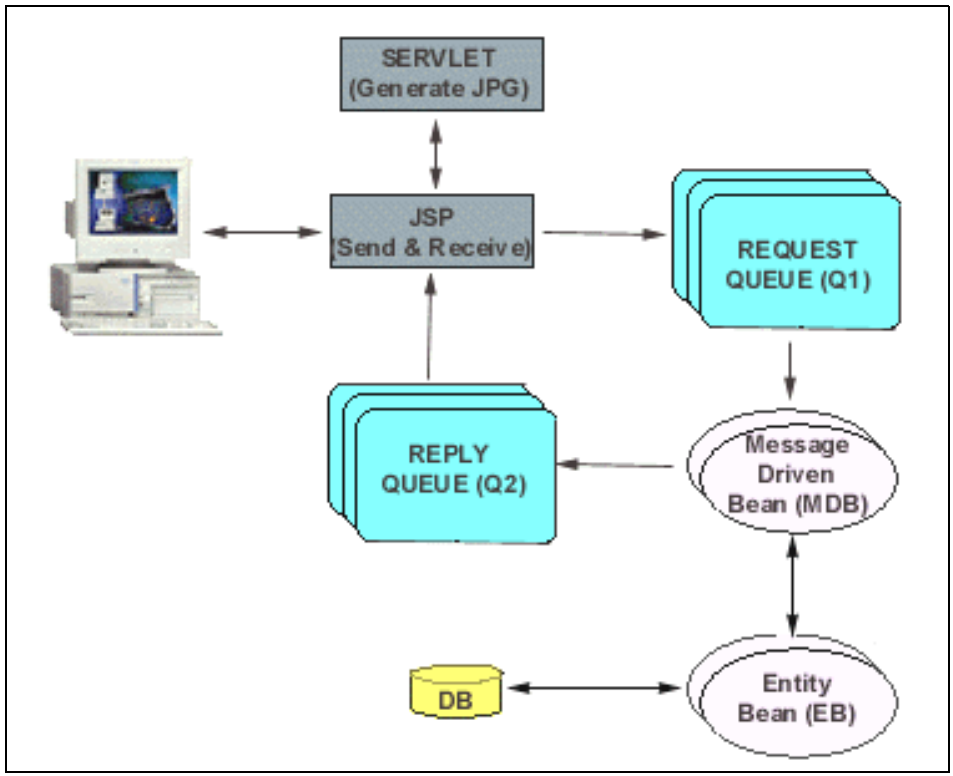


Figure 5-10 The ITSO Bank sample configuration

The ITSO Bank bean examples consists of following resources:

- Prior starting the bean sample we need to configure two request queues (LP1, LP2) and one reply queue (LP0). The listener ports LP0, LP1 and LP2 have to be assigned to the queue names Q0, Q1 and Q2 and the corresponding JNDI queue names jms/itso/Q0, jms/itso/Q1 and jms/itso/Q1. All queues adhere to the queue connection factory QCF with the JNDI name jms/itso/QCF. To define all the queue definitions use administrative console for WebSphere Application Server V5.0 or the Admin client and JMS Provider Options as described in 5.1.1, “Java Message-driven Beans” on page 60.
- **index.html in ITSOBankWeb**. In index.html there are two formulas to enter either customerid and quantity to reorder checks, or to enter a requestid to retrieve a previous check reorder. The reorder formula calls sndQ1rcvQ2.jsp with parameter ‘C=’ for the custid and ‘Q=’ for quantity and the requestid formula calls the sndQ0rcvQ2.jsp with parameter ‘R=’ for the check reorder requestid.

- **itso.bank.check.Constants.java** holds the required JNDI names, queue factory name and queue names constants used by the ITSO Bank bean sample.
- **sndQ1rcvQ2.jsp in ITSOBankWeb**. The JavaServer Page accepts the parameters 'C=' and 'Q=' and sends their values as a mapped text message to the request queue named listener port LP1 (JNDI name is jms/itso/Q1 and queue name Q1). The message driven-bean RcvQ1SndQ2Bean.java will receive this message and then call the BMP Entity Bean Check to insert one check reorder into the check_reorder table. After sending the custid-quantity mapped message to Q1, we try to receive a message from reply queue Q2 in order to check wether all components of the samples are working properly.

 Note that we limited the wait time to 1 minute (60000 m/sec) for receiving the message. If we do not a time limited receive and there is no message coming from Q2, our JSP thread would never end until there is a message on queue Q2. The following code snippet performs the receive of a message from Q2 implemented in the JSP sndQ1rcvQ2.jsp:

Example 5-21 JSP code snippet to receive a queue message using JMS

```
<%
   }
   catch (Exception e) {
      System.err.println("Error on sending message " + e);
   }
   // Receive reply message queue
   String rName =
   itso.bank.check.Constants.QUEUE_DESTINATION_JNDI_NAME_Q2;
   try {
      queue = (javax.jms.Queue) ic.lookup(rName);
   }
   catch (Exception e) {
      System.err.println("JNDI lookup failed " + e);
   }
   try {
      qConn = qConnFactory.createQueueConnection();
      qSession = qConn.createQueueSession(false,
         javax.jms.Session.AUTO_ACKNOWLEDGE);
      qReceiver = qSession.createReceiver(queue);
      qConn.start();
      // Receive the Check Reorder Message
      // during one minute
      message = (javax.jms.MapMessage)
         qReceiver.receive((long)60000);
      if (message != null) {
         ckroid = message.getInt("Requestid");
         custid =  message.getInt("Custid");
         quantity = message.getInt("Quantity");
```

```
         inserted = message.getInt("Inserted");
         customer = message.getString("Customer");
%>
      <P>Check Reorder received from ITSO Bank.<P>
      <b><% out.println("Custid: " + custid); %><br>
      <% out.println("Quantity: " + quantity); %><br>
      <% out.println("Inserted: " + inserted); %><br>
      <% out.println("Requestid: " + ckroid); %><br>
      <% out.println("Customer: " + customer); %><br></b>
<%
         System.out.println("sndQ1rcvQ2.jsp: " +
            message);
      }
      else {
         System.out.println("sndQ1rcvQ2.jsp: “ +
         “No Message received during last minute.");
         %><P>No Message could be received from ITSO Bank during last
minute.<P>
         <%
      }
      qConn.close();
   }
   catch (Exception e) {
      System.err.println("Error receiving message " + e);
   }
%>
```

- Once **sndQ1rcvQ2.jsp** receives a message from Q2, it calls the servlet **Imgmap0.java** in the itso.bank.check package of the ITSOBankWeb project as a jpeg. The Imgmap0.java servlet accepts the customer info as a parameter and produces a jpeg file with the customer info presenting the produced check to the customer as a picture.
- **sndQ0rcvQ2.jsp in ITSOBankWeb**. Instead of sending the customer-quantity message as done in sndQ1rcvQ2, this jsp sends a requestid to queue Q0. The message-driven bean RcvQ0SndQ2Bean.java will receive the message and calls the Entity Bean Check using the findByPrimaryKey() method. The results from the bean are then passed to the queue Q2 where sndQ0rcvQ2.jsp will receive the message to make the requested check reorder visible using the jpeg producer servlet Imgmap0.java.
- **itso.bank.check.Imgmap0.java in ITSOBankWeb** is a jpeg producing servlet: the contentType is “image/jpeg”. It takes the customer info with title, first name and last name, and draws it into the jpeg. The servlet imports the com.sun.image.codec.jpeg package and applies the Graphic2D package. The ITSO header.gif and the ok.jpg are also loaded into the servlet to build up the

check picture. Both pictures have to be in the same package as the servlet, because it uses the statement **this**.**getClass().getResource(name)**; to create the URL that loads both pictures.

- **jms.itso.bean.RcvQ0SndQ2Bean.java in ITSOBankEJB** is the Message-driven Bean that receives a requestid of a check reorder, previously created by the Entity Bean Check. It is also a client program for the BMP Entity Bean Check. It produces a primaryKey with the incoming requestid **ckroid**. Then the MDB bean calls the findByPrimaryKey() method in the Entity Bean Check and stores the bean's returning values of the ejbLoad() method into the value objects checkVO. To send the results back to the JSP sndQ0rcvQ2 all values in checkVO are passed to the reply queue Q2. The responsible method to send a mapped message into the reply queue Q2 is **passQ2(custid, quantity, inserted, requestid, customer)**, shown in the coding sample below:

Example 5-22 Method passQ2 in MDB RcvQ0SndQ2Bean.java

```
public class RcvQ1SndQ2Bean implements
   javax.ejb.MessageDrivenBean, javax.jms.MessageListener {
   private javax.ejb.MessageDrivenContext fMessageDrivenCtx;
   private javax.naming.Context jndiContext;
   ...

   /**
    * Pass forward to Q2: passQ2
    * Method requires jndiContext
    */
   private void passQ2(int custid, int quantity, int inserted,
      int requestid, String customer) {
      javax.jms.QueueConnectionFactory
         qConnectionFactory = null;
      javax.jms.QueueConnection qConnection = null;
      javax.jms.QueueSession qSession = null;
      javax.jms.Queue queue = null;
      javax.jms.QueueSender qSender = null;
      javax.jms.MapMessage message = null;
      try {
         qConnectionFactory =
         (javax.jms.QueueConnectionFactory)
         jndiContext.lookup(
         Constants.QUEUE_CONN_FACTORY_JNDI_NAME);
         queue = (javax.jms.Queue)
         jndiContext.lookup(
         Constants.QUEUE_DESTINATION_JNDI_NAME_Q2);
      }
      catch (javax.naming.NamingException e) {
         System.out.println(
            "RcvQ0SndQ2Bean: JNDI lookup failed: "
```

```
            + e.toString());
        }
        try {
           qConnection =
           qConnectionFactory.createQueueConnectition();
           qSession =
           qConnection.createQueueSession(false,
              javax.jms.Session.AUTO_ACKNOWLEDGE);
           qSender = qSession.createSender(queue);
           message = qSession.createMapMessage();
           message.setInt("Custid", custid);
           message.setInt("Quantity", quantity);
           message.setInt("Inserted", inserted);
           message.setInt("Requestid", requestid);
           message.setString("Customer", customer);
           qSender.send(message);
           System.out.println(
              "RcvQOSndQ2Bean: Custid: " + custid);
           System.out.println(
              "RcvQOSndQ2Bean: Quantity: "
              + quantity);
           System.out.println(
              "RcvQOSndQ2Bean: Inserted: "
              + inserted);
           System.out.println(
              "RcvQOSndQ2Bean: Requestid: "
              + requestid);
           System.out.println(
              "RcvQOSndQ2Bean: Customer: "
              + customer);
        }
        catch (javax.jms.JMSException e) {
           System.out.println(
              "RcvQOSndQ2Bean: Exception occurred: "
              + e.toString());
        }
        finally {
           if (qConnection != null) {
              try { qConnection.close(); }
              catch (javax.jms.JMSException e) {}
           }
         }
     }

}
```

- **jms.itso.bean.RcvQ1SndQ2Bean.java in ITSOBankEJB** is nearly the same as RcvQ1SndQ0Bean.java, but instead of a

home.findByPrimaryKey(primaryKey) call to the Entity Bean Check, the custid and quantity are stored as value objects of checkVO and then the home.create(checkVO) method is performed. The inserted values are returned into a new value object newcheckVO with the command bean.getCheckVO();. The value objects in newcheckVO are the passed to queue Q2 with the passQ2(int,int,intint,String) method.

- The BMP Entity Bean Check with their data access objects are applied as described in 5.1.2, “EJB 2.0 Bean Managed Persistence Entity Bean” on page 78.
- The ITSO Bank bean sample also requires the check_reorder and customer table of the ITSO Bank sample database named WSAD5.

The ITSO Bank bean sample was designed to demonstrate the development of EJB 2.0 Message-driven Beans and Bean Managed Persistence Entity Beans in WebSphere Studio Application Developer V5.0, and for the deployment in WebSphere Application Server V5.0 under Red Hat Linux Version 8.0. If there are thousands of check reorders, then we have to ensure that the messages received by the JSPs from queue Q2 are belonging to the same session ID and were initiated by the same JSP.

Otherwise, it is possible that one JSP receives a message on the queue Q2, that was born by another JSP via messages to queue Q0 or queue Q1. The sending messages do not necessarily belong to the receiving message. This problem is due to the fact, that the message producing JSPs and the message consuming EJB 2.0 Message-driven Beans have no common information; that means asynchronous messages.

The Message-driven Beans in EJB 2.0 are responsible for coordinating tasks involving other Entity Beans or session beans, The Message-driven Bean subscribes to or listens for specific asynchronous messages, to which it responds by processing the message and managing the actions of other beans. The message consumer part in the JSPs was implemented for a functional checking purpose only during development.

This approach was possible because the developer in most cases is a single user of a JSP running in the WebSphere Studio Application Developer V5.0 test server using the try-and-error method. For this situation only the approach was useful. If a reader intends to adapt the sample into a production system, we strongly recommend to eliminate the message consumer part in the JSPs and to eliminate the message forwarding part in the Message-driven Beans. The JSP then will send the check reorder requests and then forget about it. Since the messages are asynchronous, no one should expect or wait for a reply.

Running the ITSO Bank bean sample

The ITSO Bank bean sample was tested on WebSphere Studio Application Developer V5.0 and on WebSphere Application Server V5.0 on Red Hat Linux V8.0. The DB2 for Linux version we used was 7.2 with FixPak 7. If the user has created the required database tables, defined the database in his test server, and imports the ITSOBankEAR file into his V5.0 test server to run the index.html for check reorder, the following messages should appear in the console log.

Example 5-23 The ITSO Bank bean sample console log

```
[10/30/02 10:57:21:626 PST]  a1c8af4 WebGroup      I SRVE0180I: [ITSOBankWeb]
[/ITSOBankWeb] [Servlet.LOG]: /sndQ1rcvQ2.jsp: init
[10/30/02 10:57:22:728 PST] 2088cafd SystemOut     O RcvQ1SndQ2Bean:
RcvQ1SndQ2Bean()
[10/30/02 10:57:22:728 PST] 2088cafd SystemOut     O RcvQ1SndQ2Bean:
setMessageDrivenContext()
[10/30/02 10:57:22:728 PST] 2088cafd SystemOut     O RcvQ1SndQ2Bean:
ejbCreate()
[10/30/02 10:57:22:738 PST] 2088cafd SystemOut     O RcvQ1SndQ2Bean: --->
CheckBean
[10/30/02 10:57:22:838 PST] 2088cafd SystemOut     O
CheckBean:setEntityContext: START
[10/30/02 10:57:22:878 PST] 2088cafd SystemOut     O DB2CheckDAO#insertCheck:
CALLED.
[10/30/02 10:57:22:878 PST] 2088cafd SystemOut     O ConnectionPoolHelper: Look
for Datasource
[10/30/02 10:57:22:888 PST] 2088cafd ConnectionFac W J2CA0122I: Resource
reference jdbc/WSAD5 could not be located, so default values of the following
are used: [Resource-ref settings]

   res-auth:                 1 (APPLICATION)
   res-isolation-level:      0 (TRANSACTION_NONE)
   res-sharing-scope:        false (UNSHAREABLE)
   res-resolution-control:   999 (undefined)

[10/30/02 10:57:23:108 PST] 2088cafd SystemOut     O ConnectionPoolHelper: Data
Source available
[10/30/02 10:57:24:350 PST] 2088cafd SystemOut     O DB2CheckDAO#insertCheck:
111
[10/30/02 10:57:24:350 PST] 2088cafd SystemOut     O CheckBean:ejbCreate(): New
Requestid: 111
[10/30/02 10:57:24:360 PST] 2088cafd SystemOut     O DB2CheckDAO#findCustomer:
106
[10/30/02 10:57:25:362 PST] 2088cafd SystemOut     O
CheckBean:ejbCreate:CkroID=111
[10/30/02 10:57:25:362 PST] 2088cafd SystemOut     O
CheckBean:ejbCreate:CustID=106
```

```
[10/30/02 10:57:25:362 PST] 2088cafd SystemOut     O
CheckBean:ejbCreate:Quantity=6
[10/30/02 10:57:25:362 PST] 2088cafd SystemOut     O
CheckBean:ejbCreate:Status=received
[10/30/02 10:57:25:362 PST] 2088cafd SystemOut     O
CheckBean:ejbCreate:Time=null
[10/30/02 10:57:25:362 PST] 2088cafd SystemOut     O
CheckBean:ejbCreate:Customer=Mr Akis Laftsidis
[10/30/02 10:57:25:402 PST] 2088cafd SystemOut     O CheckBean:ejbPostCreate():
START
[10/30/02 10:57:25:402 PST] 2088cafd SystemOut     O CheckBean:ejbStore: START
[10/30/02 10:57:25:402 PST] 2088cafd SystemOut     O
CheckBean:ejbStore:CkroID=111
[10/30/02 10:57:25:402 PST] 2088cafd SystemOut     O
CheckBean:ejbStore:Quantity=6
[10/30/02 10:57:25:402 PST] 2088cafd SystemOut     O
CheckBean:ejbStore:Status=received
[10/30/02 10:57:25:432 PST] 2088cafd SystemOut     O CheckBean:ejbLoad: START
[10/30/02 10:57:25:432 PST] 2088cafd SystemOut     O DB2CheckDAO#getCheck: 111
[10/30/02 10:57:25:452 PST] 2088cafd SystemOut     O DB2CheckDAO#getCheck:
CkroID located.
[10/30/02 10:57:25:452 PST] 2088cafd SystemOut     O
CheckBean:ejbLoad:CkroID=111
[10/30/02 10:57:25:452 PST] 2088cafd SystemOut     O
CheckBean:ejbLoad:CustID=106
[10/30/02 10:57:25:452 PST] 2088cafd SystemOut     O
CheckBean:ejbLoad:Quantity=6
[10/30/02 10:57:25:452 PST] 2088cafd SystemOut     O
CheckBean:ejbLoad:Status=received
[10/30/02 10:57:25:452 PST] 2088cafd SystemOut     O
CheckBean:ejbLoad:Time=2002-10-30 10:57:24.34
[10/30/02 10:57:25:452 PST] 2088cafd SystemOut     O
CheckBean:ejbLoad:Customer=Mr Akis Laftsidis
[10/30/02 10:57:25:452 PST] 2088cafd SystemOut     O CheckBean:ejbStore: START
[10/30/02 10:57:25:452 PST] 2088cafd SystemOut     O
CheckBean:ejbStore:CkroID=111
[10/30/02 10:57:25:452 PST] 2088cafd SystemOut     O
CheckBean:ejbStore:Quantity=6
[10/30/02 10:57:25:452 PST] 2088cafd SystemOut     O
CheckBean:ejbStore:Status=received
[10/30/02 10:57:25:462 PST] 2088cafd SystemOut     O RcvQ1SndQ2Bean: New
CK_RO_ID: 111
[10/30/02 10:57:25:472 PST] 2088cafd SystemOut     O RcvQ1SndQ2Bean: Custid:
106
[10/30/02 10:57:25:472 PST] 2088cafd SystemOut     O RcvQ1SndQ2Bean: Quantity:
6
[10/30/02 10:57:25:472 PST] 2088cafd SystemOut     O RcvQ1SndQ2Bean: Inserted:
1
```

```
[10/30/02 10:57:25:472 PST] 2088cafd SystemOut     O RcvQ1SndQ2Bean: Requestid:
111
[10/30/02 10:57:25:472 PST] 2088cafd SystemOut     O RcvQ1SndQ2Bean: Customer:
Mr Akis Laftsidis
[10/30/02 10:57:25:482 PST]  a1c8af4 SystemOut     O sndQ1rcvQ2.jsp:
JMS Message class: jms_map
  JMSType:         null
  JMSDeliveryMode: 2
  JMSExpiration:   0
  JMSPriority:     4
  JMSMessageID:    ID:414d51205741535f6c6f63616c686f730000000000000000a
  JMSTimestamp:    1036004245472
  JMSCorrelationID:null
  JMSDestination:  queue:///WQ_Q2?expiry=0
  JMSReplyTo:      null
  JMSRedelivered:  false
  JMS_IBM_PutDate:20021030
  JMSXAppID:Application Developer
  JMS_IBM_Format:MQSTR
  JMS_IBM_PutApplType:28
  JMS_IBM_MsgType:8
  JMSXUserID:bruno
  JMS_IBM_PutTime:18572547
  JMSXDeliveryCount:1
{Custid=106, Inserted=1, Requestid=111, Quantity=6, Customer=Mr Akis Laftsidis}
[10/30/02 10:57:25:722 PST]  a1c8af4 WebGroup      I SRVE0180I: [ITSOBankWeb]
[/ITSOBankWeb] [Servlet.LOG]: ImgmapO: init
[10/30/02 10:58:06:371 PST] 3c92caf7 SystemOut     O RcvQ1SndQ2Bean: --->
CheckBean
[10/30/02 10:58:06:371 PST] 3c92caf7 SystemOut     O DB2CheckDAO#insertCheck:
CALLED.
[10/30/02 10:58:06:391 PST] 3c92caf7 SystemOut     O DB2CheckDAO#insertCheck:
112
[10/30/02 10:58:06:391 PST] 3c92caf7 SystemOut     O CheckBean:ejbCreate(): New
Requestid: 112
[10/30/02 10:58:06:391 PST] 3c92caf7 SystemOut     O DB2CheckDAO#findCustomer:
102
[10/30/02 10:58:06:641 PST] 3c92caf7 SystemOut     O
CheckBean:ejbCreate:CkroID=112
[10/30/02 10:58:06:641 PST] 3c92caf7 SystemOut     O
CheckBean:ejbCreate:CustID=102
[10/30/02 10:58:06:641 PST] 3c92caf7 SystemOut     O
CheckBean:ejbCreate:Quantity=2
[10/30/02 10:58:06:641 PST] 3c92caf7 SystemOut     O
CheckBean:ejbCreate:Status=received
[10/30/02 10:58:06:641 PST] 3c92caf7 SystemOut     O
CheckBean:ejbCreate:Time=null
[10/30/02 10:58:06:641 PST] 3c92caf7 SystemOut     O
CheckBean:ejbCreate:Customer=Mr Bruno Huelbuesch
```

```
[10/30/02 10:58:06:641 PST] 3c92caf7 SystemOut     O CheckBean:ejbPostCreate():
START
[10/30/02 10:58:06:651 PST] 3c92caf7 SystemOut     O CheckBean:ejbStore: START
[10/30/02 10:58:06:651 PST] 3c92caf7 SystemOut     O
CheckBean:ejbStore:CkroID=112
[10/30/02 10:58:06:651 PST] 3c92caf7 SystemOut     O
CheckBean:ejbStore:Quantity=2
[10/30/02 10:58:06:651 PST] 3c92caf7 SystemOut     O
CheckBean:ejbStore:Status=received
[10/30/02 10:58:06:651 PST] 3c92caf7 SystemOut     O CheckBean:ejbLoad: START
[10/30/02 10:58:06:651 PST] 3c92caf7 SystemOut     O DB2CheckDAO#getCheck: 112
[10/30/02 10:58:06:651 PST] 3c92caf7 SystemOut     O DB2CheckDAO#getCheck:
CkroID located.
[10/30/02 10:58:06:651 PST] 3c92caf7 SystemOut     O
CheckBean:ejbLoad:CkroID=112
[10/30/02 10:58:06:651 PST] 3c92caf7 SystemOut     O
CheckBean:ejbLoad:CustID=102
[10/30/02 10:58:06:651 PST] 3c92caf7 SystemOut     O
CheckBean:ejbLoad:Quantity=2
[10/30/02 10:58:06:651 PST] 3c92caf7 SystemOut     O
CheckBean:ejbLoad:Status=received
[10/30/02 10:58:06:651 PST] 3c92caf7 SystemOut     O
CheckBean:ejbLoad:Time=2002-10-30 10:58:06.38
[10/30/02 10:58:06:651 PST] 3c92caf7 SystemOut     O
CheckBean:ejbLoad:Customer=Mr Bruno Huelbuesch
[10/30/02 10:58:06:651 PST] 3c92caf7 SystemOut     O CheckBean:ejbStore: START
[10/30/02 10:58:06:651 PST] 3c92caf7 SystemOut     O
CheckBean:ejbStore:CkroID=112
[10/30/02 10:58:06:651 PST] 3c92caf7 SystemOut     O
CheckBean:ejbStore:Quantity=2
[10/30/02 10:58:06:651 PST] 3c92caf7 SystemOut     O
CheckBean:ejbStore:Status=received
[10/30/02 10:58:06:651 PST] 3c92caf7 SystemOut     O RcvQ1SndQ2Bean: New
CK_RO_ID: 112
[10/30/02 10:58:06:661 PST] 3c92caf7 SystemOut     O RcvQ1SndQ2Bean: Custid:
102
[10/30/02 10:58:06:661 PST] 3c92caf7 SystemOut     O RcvQ1SndQ2Bean: Quantity:
2
[10/30/02 10:58:06:661 PST] 3c92caf7 SystemOut     O RcvQ1SndQ2Bean: Inserted:
1
[10/30/02 10:58:06:661 PST] 3c92caf7 SystemOut     O RcvQ1SndQ2Bean: Requestid:
112
[10/30/02 10:58:06:661 PST] 3c92caf7 SystemOut     O RcvQ1SndQ2Bean: Customer:
Mr Bruno Huelbuesch
[10/30/02 10:58:06:671 PST] 3c91caf7 SystemOut     O sndQ1rcvQ2.jsp:
JMS Message class: jms_map
  JMSType:         null
  JMSDeliveryMode: 2
  JMSExpiration:   0
```

```
  JMSPriority:     4
  JMSMessageID:    ID:414d51205741535f6c6f63616c686f73000000000000000c
  JMSTimestamp:    1036004286661
  JMSCorrelationID:null
  JMSDestination:  queue:///WQ_Q2?expiry=0
  JMSReplyTo:      null
  JMSRedelivered:  false
  JMS_IBM_PutDate:20021030
  JMSXAppID:Application Developer
  JMS_IBM_Format:MQSTR
  JMS_IBM_PutApplType:28
  JMS_IBM_MsgType:8
  JMSXUserID:bruno
  JMS_IBM_PutTime:18580666
  JMSXDeliveryCount:1
{Custid=102, Inserted=1, Requestid=112, Quantity=2, Customer=Mr Bruno
Huelbuesch}
[10/30/02 10:58:40:279 PST] 3c97caf7 WebGroup      I SRVE0180I: [ITSOBankWeb]
[/ITSOBankWeb] [Servlet.LOG]: /sndQOrcvQ2.jsp: init
[10/30/02 10:58:40:370 PST] 3ca48af7 SystemOut     O RcvQOSndQ2Bean:
RcvQOSndQ2Bean()
[10/30/02 10:58:40:370 PST] 3ca48af7 SystemOut     O RcvQOSndQ2Bean:
setMessageDrivenContext()
[10/30/02 10:58:40:370 PST] 3ca48af7 SystemOut     O RcvQOSndQ2Bean:
ejbCreate()
[10/30/02 10:58:40:370 PST] 3ca48af7 SystemOut     O RcvQOSndQ2Bean: --->
CheckBean
[10/30/02 10:58:40:380 PST] 3ca48af7 SystemOut     O
CheckKey:ejbFindByPrimaryKey: START
[10/30/02 10:58:40:380 PST] 3ca48af7 SystemOut     O DB2CheckDAO#findCheck: 111
[10/30/02 10:58:40:390 PST] 3ca48af7 SystemOut     O DB2CheckDAO#findCheck:
CkroID located.
[10/30/02 10:58:40:390 PST] 3ca48af7 SystemOut     O
CheckBean:setEntityContext: START
[10/30/02 10:58:40:390 PST] 3ca48af7 SystemOut     O CheckBean:ejbLoad: START
[10/30/02 10:58:40:390 PST] 3ca48af7 SystemOut     O DB2CheckDAO#getCheck: 111
[10/30/02 10:58:40:890 PST] 3ca48af7 SystemOut     O DB2CheckDAO#getCheck:
CkroID located.
[10/30/02 10:58:40:890 PST] 3ca48af7 SystemOut     O
CheckBean:ejbLoad:CkroID=111
[10/30/02 10:58:40:900 PST] 3ca48af7 SystemOut     O
CheckBean:ejbLoad:CustID=106
[10/30/02 10:58:40:900 PST] 3ca48af7 SystemOut     O
CheckBean:ejbLoad:Quantity=6
[10/30/02 10:58:40:900 PST] 3ca48af7 SystemOut     O
CheckBean:ejbLoad:Status=received
[10/30/02 10:58:40:900 PST] 3ca48af7 SystemOut     O
CheckBean:ejbLoad:Time=2002-10-30 10:57:24.34
```

```
[10/30/02 10:58:40:900 PST] 3ca48af7 SystemOut     O
CheckBean:ejbLoad:Customer=Mr Akis Laftsidis
[10/30/02 10:58:40:900 PST] 3ca48af7 SystemOut     O CheckBean:ejbLoad: START
[10/30/02 10:58:40:900 PST] 3ca48af7 SystemOut     O DB2CheckDAO#getCheck: 111
[10/30/02 10:58:40:900 PST] 3ca48af7 SystemOut     O DB2CheckDAO#getCheck:
CkroID located.
[10/30/02 10:58:40:900 PST] 3ca48af7 SystemOut     O
CheckBean:ejbLoad:CkroID=111
[10/30/02 10:58:40:900 PST] 3ca48af7 SystemOut     O
CheckBean:ejbLoad:CustID=106
[10/30/02 10:58:40:900 PST] 3ca48af7 SystemOut     O
CheckBean:ejbLoad:Quantity=6
[10/30/02 10:58:40:900 PST] 3ca48af7 SystemOut     O
CheckBean:ejbLoad:Status=received
[10/30/02 10:58:40:900 PST] 3ca48af7 SystemOut     O
CheckBean:ejbLoad:Time=2002-10-30 10:57:24.34
[10/30/02 10:58:40:900 PST] 3ca48af7 SystemOut     O
CheckBean:ejbLoad:Customer=Mr Akis Laftsidis
[10/30/02 10:58:40:900 PST] 3ca48af7 SystemOut     O CheckBean:ejbStore: START
[10/30/02 10:58:40:900 PST] 3ca48af7 SystemOut     O
CheckBean:ejbStore:CkroID=111
[10/30/02 10:58:40:990 PST] 3ca48af7 SystemOut     O
CheckBean:ejbStore:Quantity=6
[10/30/02 10:58:40:990 PST] 3ca48af7 SystemOut     O
CheckBean:ejbStore:Status=received
[10/30/02 10:58:41:000 PST] 3ca48af7 SystemOut     O RcvQOSndQ2Bean: Custid:
106
[10/30/02 10:58:41:000 PST] 3ca48af7 SystemOut     O RcvQOSndQ2Bean: Quantity:
6
[10/30/02 10:58:41:000 PST] 3ca48af7 SystemOut     O RcvQOSndQ2Bean: Inserted:
0
[10/30/02 10:58:41:000 PST] 3ca48af7 SystemOut     O RcvQOSndQ2Bean: Requestid:
111
[10/30/02 10:58:41:000 PST] 3ca48af7 SystemOut     O RcvQOSndQ2Bean: Customer:
Mr Akis Laftsidis
[10/30/02 10:58:41:010 PST] 3c97caf7 SystemOut     O sndQOrcvQ2.jsp:
JMS Message class: jms_map
  JMSType:         null
  JMSDeliveryMode: 2
  JMSExpiration:   0
  JMSPriority:     4
  JMSMessageID:    ID:414d51205741535f6c6f63616c686f730000000000000000e
  JMSTimestamp:    1036004321000
  JMSCorrelationID:null
  JMSDestination:  queue:///WQ_Q2?expiry=0
  JMSReplyTo:      null
  JMSRedelivered:  false
  JMS_IBM_PutDate:20021030
  JMSXAppID:Application Developer
```

```
  JMS_IBM_Format:MQSTR
  JMS_IBM_PutApplType:28
  JMS_IBM_MsgType:8
  JMSXUserID:bruno
  JMS_IBM_PutTime:18584100
  JMSXDeliveryCount:1
{Custid=106, Inserted=0, Requestid=111, Quantity=6, Customer=Mr Akis Laftsidis}
[10/30/02 10:58:49:052 PST] 3caa8af7 SystemOut     O DB2CheckDAO#getCheck:
CkroID located.
[10/30/02 10:58:49:052 PST] 3caa8af7 SystemOut     O
CheckBean:ejbLoad:CkroID=112
[10/30/02 10:58:49:052 PST] 3caa8af7 SystemOut     O
CheckBean:ejbLoad:CustID=102
[10/30/02 10:58:49:052 PST] 3caa8af7 SystemOut     O
CheckBean:ejbLoad:Quantity=2
[10/30/02 10:58:49:052 PST] 3caa8af7 SystemOut     O
CheckBean:ejbLoad:Status=received
[10/30/02 10:58:49:052 PST] 3caa8af7 SystemOut     O
CheckBean:ejbLoad:Time=2002-10-30 10:58:06.38
[10/30/02 10:58:49:052 PST] 3caa8af7 SystemOut     O
CheckBean:ejbLoad:Customer=Mr Bruno Huelbuesch
[10/30/02 10:58:49:062 PST] 3caa8af7 SystemOut     O CheckBean:ejbLoad: START
[10/30/02 10:58:49:062 PST] 3caa8af7 SystemOut     O DB2CheckDAO#getCheck: 112
[10/30/02 10:58:49:062 PST] 3caa8af7 SystemOut     O DB2CheckDAO#getCheck:
CkroID located.
[10/30/02 10:58:49:062 PST] 3caa8af7 SystemOut     O
CheckBean:ejbLoad:CkroID=112
[10/30/02 10:58:49:062 PST] 3caa8af7 SystemOut     O
CheckBean:ejbLoad:CustID=102
[10/30/02 10:58:49:062 PST] 3caa8af7 SystemOut     O
CheckBean:ejbLoad:Quantity=2
[10/30/02 10:58:49:062 PST] 3caa8af7 SystemOut     O
CheckBean:ejbLoad:Status=received
[10/30/02 10:58:49:062 PST] 3caa8af7 SystemOut     O
CheckBean:ejbLoad:Time=2002-10-30 10:58:06.38
[10/30/02 10:58:49:062 PST] 3caa8af7 SystemOut     O
CheckBean:ejbLoad:Customer=Mr Bruno Huelbuesch
[10/30/02 10:58:49:072 PST] 3caa8af7 SystemOut     O CheckBean:ejbStore: START
[10/30/02 10:58:49:072 PST] 3caa8af7 SystemOut     O
CheckBean:ejbStore:CkroID=112
[10/30/02 10:58:49:082 PST] 3caa8af7 SystemOut     O
CheckBean:ejbStore:Quantity=2
[10/30/02 10:58:49:082 PST] 3caa8af7 SystemOut     O
CheckBean:ejbStore:Status=received
[10/30/02 10:58:49:082 PST] 3caa8af7 SystemOut     O RcvQOSndQ2Bean: Custid:
102
[10/30/02 10:58:49:082 PST] 3caa8af7 SystemOut     O RcvQOSndQ2Bean: Quantity:
2
```

```
[10/30/02 10:58:49:082 PST] 3caa8af7 SystemOut     O RcvQOSndQ2Bean: Inserted:
0
[10/30/02 10:58:49:082 PST] 3caa8af7 SystemOut     O RcvQOSndQ2Bean: Requestid:
112
[10/30/02 10:58:49:092 PST] 3caa8af7 SystemOut     O RcvQOSndQ2Bean: Customer:
Mr Bruno Huelbuesch
[10/30/02 10:58:49:092 PST] 3ca98af7 SystemOut     O sndQOrcvQ2.jsp:
JMS Message class: jms_map
  JMSType:          null
  JMSDeliveryMode: 2
  JMSExpiration:   0
  JMSPriority:     4
  JMSMessageID:    ID:414d51205741535f6c6f63616c686f730000000000000010
  JMSTimestamp:    1036004329092
  JMSCorrelationID:null
  JMSDestination:  queue:///WQ_Q2?expiry=0
  JMSReplyTo:      null
  JMSRedelivered:  false
  JMS_IBM_PutDate:20021030
  JMSXAppID:Application Developer
  JMS_IBM_Format:MQSTR
  JMS_IBM_PutApplType:28
  JMS_IBM_MsgType:8
  JMSXUserID:bruno
  JMS_IBM_PutTime:18584909
  JMSXDeliveryCount:1
{Custid=102, Inserted=0, Requestid=112, Quantity=2, Customer=Mr Bruno
Huelbuesch}
```

6

The eXtensive Markup Language

In this chapter, we discuss the following sections:

- XML tools in WebSphere Application Developer gives an introduction to the XML tools that are available in WebSphere Application Developer for Linux Version 5.
- Introducing ITSO Banking example using XML gives an introduction to the ITSO Banking example as used in this chapter.
- Using the wizards to create XML from SQL shows how to create and generate XML from an SQL query. Also, describes how to use the various editors to update XML files.
- Dynamically generating the XML from SQL describes the use of “SQLtoXML” class library with the example servlet.
- Using the XSL debugger and transformation tools shows how the XSL debugger and transformation tool helps you debug or transform XSL stylesheets.
- Motivation to use XML/XSL instead of JSP gives a short opportunity to use XML/XSL with the XSLT technology instead of JavaServer Pages in a Web application using Struts.

© Copyright IBM Corp. 2003. All rights reserved.

6.1 XML tools in WebSphere Application Developer

Following is a list of the visual XML development environments in WebSphere Studio Application Developer V5 for Linux:

- Debug and edit XSL files with code assist
- Create, view and validate DTDs, XML schemas, XML documents and XSL style sheets
- Generate JavaBeans from XML
- XML security contains a new digital signature wizard
- Java XML/XSL wizard client that help you convert JavaBean data into a DOM tree
- Generate XSL script that transform document that are defined mappings between XML documents
- Generate an HTML or XML documents by applying an XSL stylesheet to an XML document using the Xalan processor
- Produce XML from an SQL query
- Build mappings between DTD files and relational tables
- Generate a document access definition (DAD) script, used by IBM DB2 XML Extender, to compose XML documents from existing DB2 data, or decompose XML documents into DB2 data
- Create and execute XPath using the wizard

The following list are the new functions in WebSphere Studio Application Developer Version 5:

- New XSL Debug Perspective to step visually through an XSL transformation script, highlighting the transformation. Debugging XSLs in now easy!
- The powerful XSL wizards make creation of XSL stylesheets easy!
- New XSL Editor - XPath wizard creates visually and tests XPath expressions before adding to an XSL document.
- Java Bean XML/XSL Client Wizard - Converts the data in a Java Bean (including EJBs) into a DOM tree
- XML Security - Uses the advanced encryption technology for securing XML documents
- XML Editor enhancements to associate XSL/DTD/XSD with an XML document. Simplifies document validation and transformation.

6.2 Introducing ITSO Banking example using XML

In this chapter, we will describe some of the features that we use for our example, and give you an overview of the various parts of the ITSO Banking example. Walking through our example will give you an opportunity to get started with the new XML features that are available for WebSphere Studio Application Developer on Linux.

The following list includes some of the features that we cover:

- How to build an XML from an SQL query - The SQL to XML wizard creates an XML source code from a query. You can create an XML schema or a DTD that will build the structure of your XML source code. There are two Java class libraries, *SQLtoXML* and *XMLtoSQL,* which are required for your applications to execute successfully. For example, in our example we use the SQLtoXML Java class library to perform the database queries.
- Creating HTML or XML documents by applying an XSL stylesheet against an XML document using the Xalan processor. Our example uses the Xalan processor to build the HTML and XML documents.
- XSL debugging and transformation tools - This allows you apply XSL documents to XML documents that transform them into new HTML documents. You may use the XSL debugger to visually step through an XSL transformation script.
- Show the use of the various XML editors - The XML editor can be used to inspect, modify, and validate XML documents. The DTD editor can be used to view DTD documents and also convert them into XSD (XML schema) documents. The editors are described in details later in the chapter.
- Use of user DTDs, XML schemas, XML documents, and XSL stylesheets

The ITSO Banking example for this chapter will perform the following functions:

- Acquires the user ID and password
- Authenticating the user
- Performs the necessary calculations to obtain the account balance for the user; uses an SQL query to access the example WSAD5 database
- Displays the results in an HTML page

Once you have completed this chapter, the SQL statement will be as follows (Example 6-1).

Example 6-1 Select statement for customer account balance

```
SELECT
   ITSO.CUSTOMER.FIRSTNAME,
   ITSO.CUSTOMER.LASTNAME,
   ITSO.ACCOUNT.BALANCE,
   ITSO.ACCOUNT.ACCTYPE
FROM
   ITSO.CUSTOMER, ITSO.CUSTACCT, ITSO.ACCOUNT
WHERE
   ITSO.CUSTOMER.CUSTOMERID = ITSO.CUSTACCT.CUSTOMERID
   AND ITSO.CUSTACCT.ACCID = ITSO.ACCOUNT.ACCID
   AND ITSO.CUSTOMER.USERID = :userid
   AND ITSO.CUSTOMER.PASSWORD = :passwd
```

The ITSO Banking example functions are built using XML not Enterprise JavaBeans as shown in Chapter 5, “Enterprise JavaBeans 2.0” on page 59. As stated earlier, there are two Java class libraries, *SQLtoXML* and *XMLtoSQL*, which are required for your applications to execute successfully. Since our SQL query uses select, we will mainly be using the SQLtoXML library. The XMLtoSQL library is used for inserts or deletes. The functions for the example are created in two stages:

- Using the wizards to create XML from SQL queries. We use the RDB to XML mapping (SQL to XML wizard) editor to execute a database query and convert the results into an XML format. During all this steps, the mapping editor generates the DTD or the XML schema describing a list of customers and their balances.
- Dynamically generating the XML from SQL queries. We go to implement the dynamic way from SQL to XML to use the SQLtoXML library to get the dynamical results. The SQLtoXML is a library. This library provides useful classes/methods that can be used by servlets and other development tools (SQL to XML wizard) to perform database queries. The query result is to obtain in an XML format corresponding DTD, XML schema, and a default XSL to transform the generated XML data into HTML.

We have also added some information to help you use the XSL debugger and transformation tools. At the end of the XML chapter, you may find that the XML/XSL development is similar to JavaServer Pages (JSP) and it shows an opportunity to develop a Web application without JSP technology (using the StrutsCX Framework).

6.3 Using the wizards to create XML from SQL

In the following section, we will show the database to XML mapping, how to create an SQL query, generating XML from the SQL query, and how to use the XML, DTD, and XSL editors.

6.3.1 RDB to XML mapping

In data perspective of your Workbench, you will find the SQL to XML wizard that helps you produce a SQL statement. The SQL to XML wizard gives the developers the capability of generating XSL and XML files from a SQL statement.

The Figure 6-1 shows the flow of the RDB to XML mapping as generated by the wizard.

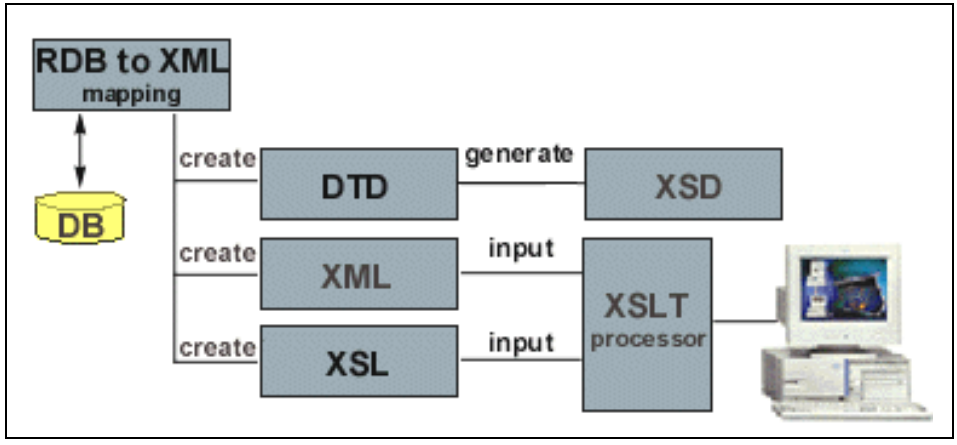


Figure 6-1 Flow of the RDB to XML mapping

The above figure clearly shows that the wizard will create the necessary documents as well as a frontend HTML. The select statement is shown in Example 6-1 on page 116.

Here are the steps that you have to follow to create an RDB to XML mapping:

1. Create a database connection.
2. Import the connection to your project.
3. Use the SQL builder to build the query.
4. Use the RDB to XML mapping to convert the results into XML.

6.3.2 Create a SQL query

In this section we will describe steps 1, 2, and 3 for creating RDB to XML mapping:

1. Create an XML workspace by issuing the following command from your home directory:

   ```
   wsappdev50 -data workspace_XML
   ```

2. Change to the Data perspective.
3. In the DB Servers window, right-click **WSAD5** and select **Reconnect**. If you need to re-connect to the database use the following variables:

 Connection name: WSAD5
 Database name: WSAD5
 UserID: db2inst1
 Password: ********
 JDBC driver class is: db2java.zip

4. Expand the **WSAD5** connection and select **WSAD(jdbc:db2:WSAD5)** database.
5. Right-click and from the pop-up menu select **Import to Folder**.
6. Click **Browse** and select the **ITSOBankData**. This folder does not exist, therefore, it will be automatically created by the tool.
7. In the **Data Definition** view expand **ITSOBankData** project and navigate to the statement folder located under **/Web Content/WEB-INF/databases.** Create a new select statement called **customerBalance**. Enter the select query form, replace the default *SELECT ** query and save the editor contents.
8. To execute the query click this button: 
9. Enter the variable as shown in Figure 6-2.

Name	Type	Value
:userid	CHARACTER	'cust101'
:passwd	CHARACTER	'ws'

Figure 6-2 Input of specify variables

You should get the following results (Figure 6-3).

FIRSTNAME	LASTNAME	BALANCE	ACCTYPE
Wolfgang	Sanyer	10000.00	CHECKING
Wolfgang	Sanyer	98726.26	SAVINGS
Wolfgang	Sanyer	5726.26	SAVINGS

Figure 6-3 Result of the SQL query in the SQL builder

You are now ready to move to the next step.

6.3.3 Generate XML from SQL query

In this section we will describe steps 4 for creating RDB to XML mapping:

1. It is very simple to create the transformation files from SQL to XML, however, before we begin we have to create a Web application that contains an Enterprise Application. Use the Enterprise Application wizard to create a Enterprise Application Project called *ITSOBankEAR* and associate this with a new Web project called *ITSOBankWeb*. You will now have two new projects.
2. Expand the **ITSOBankWeb** project under **libraries** in the Navigator view. Ensure that the sqltoxml.jar library is available. You can acquire this from the following location in the file system:

 `/opt/IBM/WebSphereStudio/eclipse/plugins/com.ibm.etools.sqltoxml/jar`
3. If required, import this to the *ITSOBankWeb* project.
4. Open the **Data Definition** perspective, select the **customerBalance** statement, right-click, and from the pop-up menu select **Generate new XML.**
5. In the **XML from An SQL Query** window verify that the information you have supplied in this wizard is exactly as shown in Figure 6-4.

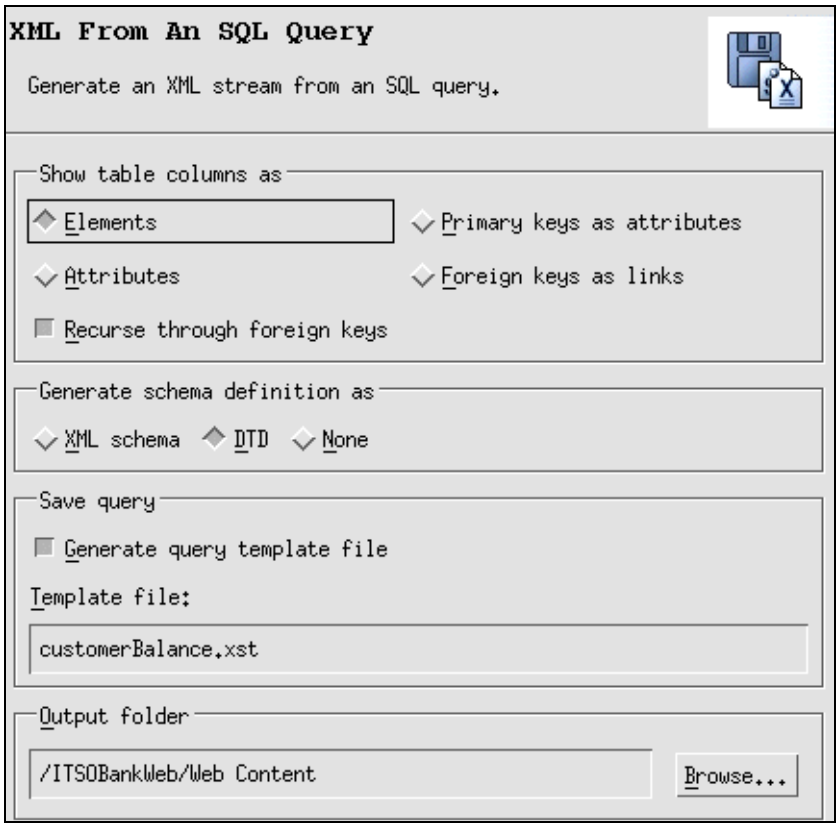


Figure 6-4 XML From An SQL Query Wizard

6. You may need to specify the user ID and password. To indicate them, use single quotes around the values to indicate values for the user ID and password (see Figure 6-2 on page 118).
7. Open the **Web Perspective** and select the **Navigator** view. Expand the **Web Content** directory as follows (Figure 6-5).

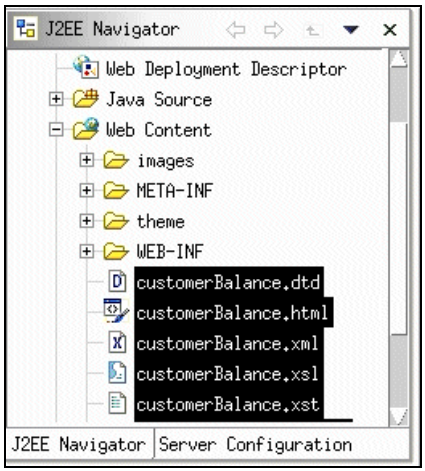


Figure 6-5 Generated files for customerBalance example

You will notice that five different files have been created for customerBalance. Each of these files have different extensions. You have successfully generated an RDB to XML mapping.

In the next section, we will show that the editors can be used to view or update these files, as well as the contents of each file:

6.3.4 XML, DTD, and XSL editors

The following subsections describes with generated output files the opportunities when using the editors in WebSphere Studio Application Developer.

XML editor

Switch to the XML Perspective, in the Navigator view, double-click **customerBalance.xml** file to open with the XML editor. Expand the document structure in the **Design** view. Click the **Source** tab. The Source view of the XML editor shows the results of the query. The following are the results of the query in XML format. The XML file was generated by the SQLToXML class, which acquires input, *userid=cust101* and *passwd=ws*, from the customerBalance.xst file.

Example 6-2 SQL results are stored in the customerBalance.xml file

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE SQLResult SYSTEM "customerBalance.dtd">
<SQLResult>
    <CUSTOMER_CUSTACCT_ACCOUNT>
```

```
        <FIRSTNAME>Wolfgang</FIRSTNAME>
        <LASTNAME>Sanyer</LASTNAME>
        <BALANCE>10000.00</BALANCE>
        <ACCTYPE>CHECKING</ACCTYPE>
    </CUSTOMER_CUSTACCT_ACCOUNT>
    <CUSTOMER_CUSTACCT_ACCOUNT>
        <FIRSTNAME>Wolfgang</FIRSTNAME>
        <LASTNAME>Sanyer</LASTNAME>
        <BALANCE>98726.26</BALANCE>
        <ACCTYPE>SAVINGS</ACCTYPE>
    </CUSTOMER_CUSTACCT_ACCOUNT>
    <CUSTOMER_CUSTACCT_ACCOUNT>
        <FIRSTNAME>Wolfgang</FIRSTNAME>
        <LASTNAME>Sanyer</LASTNAME>
        <BALANCE>5726.26</BALANCE>
        <ACCTYPE>SAVINGS</ACCTYPE>
    </CUSTOMER_CUSTACCT_ACCOUNT>
</SQLResult>
```

DTD editor

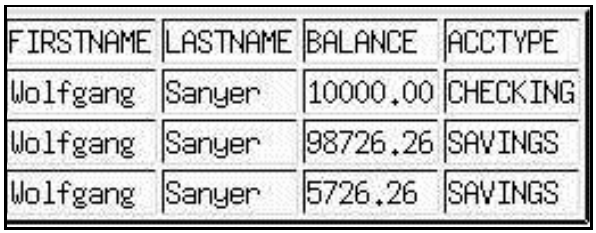
In the XML Perspective double-click the **customerBalance.dtd** file in the Navigator view. This file contains the Document Type Definition (DTD) used by the query result file customerBalance.xml. It contains the declarations that define elements for a particular XML file and establishes constraints for how each element may be used within that particular XML file.

You can use the Outline view to add or remove components of your DTD. When you select an object in this view, the Design view will display the properties that are associated with the DTD component object. Also, you can switch to the Source view to browse and edit the DTD source directly. The Task view is also active from the Workbench for reporting errors in the DTD file.

XSL editor

This is a text editor to view the source code of an XSL file. It has several text editing features like content assistant and syntax highlighting. Another useful feature of this editor is the incremental validation. At any point during your development of an XSL file, you can start the validation process to validate the file. Just right-click on the **file name** in the Navigator view, and select **Validate XSL File**. The validation is now running and any error will be reported in red in the Task view. The validation process also runs when you save a document. To view the file in the XML Perspective under the Navigator view, double-click **customerBalance.xsl**. This has an XML stylesheet that describes XML to HTML conversion. The XSL file defines the layout of your Web page.

The customerBalance.html file is a sample HTML file created when the customerBalance.xsl transformation was applied to customerBalance.xml. The wizard calls the SQLtoXML library with the required parameters. You can open the file to see the formatted results from the XML perspective. The HTML output is as follows (Figure 6-6).



FIRSTNAME	LASTNAME	BALANCE	ACCTYPE
Wolfgang	Sanyer	10000.00	CHECKING
Wolfgang	Sanyer	98726.26	SAVINGS
Wolfgang	Sanyer	5726.26	SAVINGS

Figure 6-6 customerBalance.html

Example 6-3 is the content of customerBalance.xst that contains information about the database and the query.

Example 6-3 File customerBalance.xst with database information

```
<?xml version="1.0" encoding="UTF-8"?>
<SQLGENERATEINFORMATION>
  <DATABASEINFORMATION>
    <LOGINID>db2admin</LOGINID>
    <PASSWORD><![CDATA[db2admin]]></PASSWORD>
    <JDBCDRIVER>COM.ibm.db2.jdbc.app.DB2Driver</JDBCDRIVER>
    <JDBCSERVER>jdbc:db2:WSAD5</JDBCSERVER>
  </DATABASEINFORMATION>
  <STATEMENT>
    <![CDATA[ SELECT ITSO.CUSTOMER.FIRSTNAME, ITSO.CUSTOMER.LASTNAME,
ITSO.ACCOUNT.BALANCE, ITSO.ACCOUNT.ACCTYPE FROM ITSO.CUSTOMER, ITSO.CUSTACCT,
ITSO.ACCOUNT WHERE ITSO.CUSTOMER.CUSTOMERID = ITSO.CUSTACCT.CUSTOMERID AND
ITSO.CUSTACCT.ACCID = ITSO.ACCOUNT.ACCID AND ITSO.CUSTOMER.USERID = :userid AND
ITSO.CUSTOMER.PASSWORD = :passwd  ]]>
  </STATEMENT>
  <OPTIONS>
    <FORMATOPTION>GENERATE_AS_ELEMENTS</FORMATOPTION>
    <RECURSE>FALSE</RECURSE>
  </OPTIONS>
</SQLGENERATEINFORMATION>
```

In the next section we will use *dynamically way* using XML from SQL now that the XML results file will not be stored on the disk. It will be generated in the memory.

6.4 Dynamically generating XML from SQL

In this section we want to dynamically generate XML from the SQL query at runtime using the **SQLtoXML** class library within a servlet. The library works with all JDBC databases, such as DB2, Oracle, or Sybase.

The Figure 6-7 shows that the servlet calls to the SQLtoXML class library and transforms it into XML document (DOM Object), then by using the XSLT processor to transform to simple HTML.

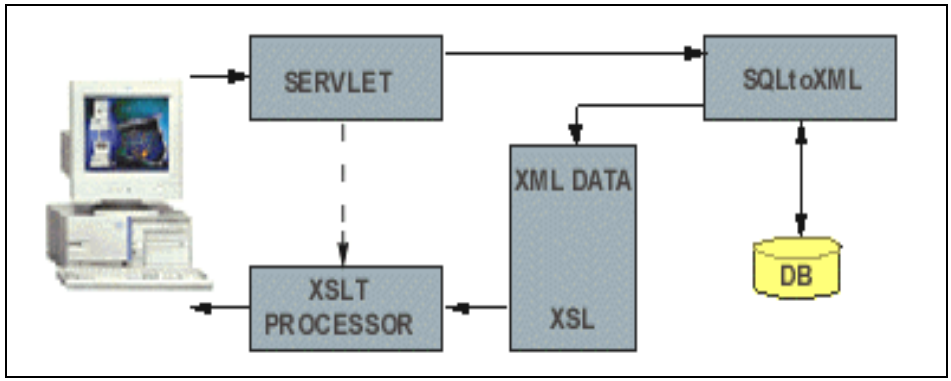


Figure 6-7 Servlet access to get XML document

The following steps describe more detail:

1. The HMTL page sends a form with user ID and password to the servlet via HTTP request, invoking the servlet doPost() method.
2. The servlet filters the user ID and password from the HTTP request and invokes the SQLToXML class library with two parameters.
3. The SQLToXML class library starts the access to the database and retrieves the data from the database like an XML document.

> **Note:** The result of the SQLToXML library in WebSphere Studio Application Developer Version 5 returns a XML document back. Earlier versions of WebSphere Studio return the result as stream data. Now no parsing is necessary.

4. The servlet gets an XML document from the SQLToXML library and instantiates the XSLT Processor
5. The XSL stylesheet is applied and the HTML result page has been generated and transferred to the browser. The user can see his actual account balances.

6.4.1 Setting up a Web project

To set up the working environment, import files from the samples\ch09 folder into your Web project (ITSOBankWeb). Figure 6-8 is a Navigator view of the Web project and its contents.

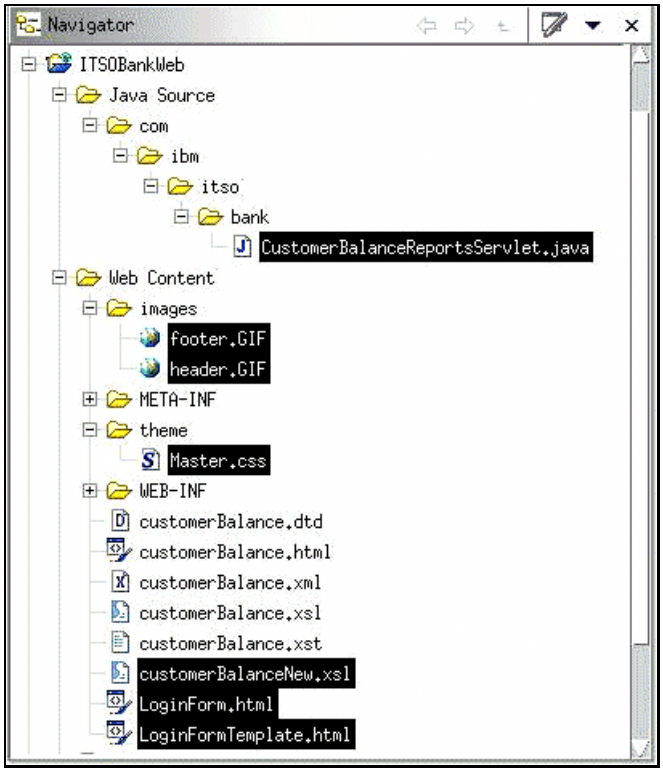


Figure 6-8 Setting up the environment

You will also need to modify the Deployment Descriptor of the Web project. In the web.xml file, the servlet tags should contain the following information (Example 6-4).

Example 6-4 Coding to add in the Deployment Descriptor web.xml

```
<servlet>
<servlet-name>CustomerBalanceReportsServlet</servlet-name>
<display-name>CustomerBalanceReportsServlet</display-name>
<servlet-class>com.ibm.itso.bank.CustomerBalanceReportsServlet</servlet-class>
</servlet>
<servlet-mapping>
<servlet-name>CustomerBalanceReportsServlet</servlet-name>
<url-pattern>/CustomerBalanceReportsServlet</url-pattern>
```

```
</servlet-mapping>
```

After these additional changes, you have to rebuild the Web project.

6.4.2 Walking through the Web application

Before we execute the Web project, we will explain some of the key files that are used in the project:

- **LoginForm.html -** This is a form where you enter the user ID and password of the customer to view the account balances. The **Submit** button calls a servlet and the HTTP request sends parameters.

  ```
  <FORM METHOD="post"ACTION="CustomerBalanceReportsServlet">
  ```

- Following is the init()-method of the servlet, it will create one instance of the servlet to read the database information from the CustomerBalance.xst file to set the database information in a QueryProperty. We also instantiate SQLToXML in this method (see Example 6-5).

Example 6-5 Init - method of the CustomerBalanceReportsServlet

```
public void init(ServletConfig config) throws ServletException
{
   super.init(config);
   qprops = new QueryProperties();
   try {
   URL url = getServletContext().getResource("customerBalance.xst");
   DataInputStream dis = new DataInputStream(url.openStream());
   qprops.load(dis);
   sql2xml = new SQLToXML(qprops);
   }catch (Exception e) {}
}
```

- The doPost()-method filters the user ID and password with the getparameter()- method as input parameter values into the instance of the SQLToXML class. After this, the SQLToXML class will be executed to get XML document as a result. The getXSL()- method helps to read the CustomerBalance.xsl file. To generate HTML, the DOM object and the xsl variable will be used as input parameter values of the genHTML()- method to send HTML via the HTTP response back to the browser (see Example 6-6).

Example 6-6 doPost()- method of CustomerBalanceReportsServlet

```
public void doPost(HttpServletRequest req, HttpServletResponse res)
       throws IOException {
       res.setContentType("text/html");
       resp = res.getWriter();
       String userid = getParameter(req, "userid");
```

```
        String passwd = getParameter(req, "passw");
        try {
            sql2xml.setParameters("'" + userid + "','" + passwd + "'");
            sql2xml.execute();
            String xsl = getXSL(req);
            genHTML(xsl,res,sql2xml.getCurrentDocument());
        } catch (Exception e) {
            error(resp, e);
        }
    }
```

The genHTML()- method represented the transformation as follows:

1. Get an instance of the TransformerFactory by calling its newInstance()- method.
2. Get a transformer from the TransformerFactory by calling the XSL source's newTransformer()- method.
3. Call the transform()- method by referencing a new DOMSource instance as source and a StreamResult instance as result to write HTML back.

Example 6-7 XSLT transformation

```
private void genHTML(String xsl, HttpServletResponse res, Document doc) throws
Exception {
       res.setContentType("text/html");
       PrintWriter writer = res.getWriter();
       TransformerFactory tFactory = TransformerFactory.newInstance();
       Transformer transformer = tFactory.newTransformer(new StreamSource(new
StringReader(xsl)));
       transformer.transform(new DOMSource(doc),new StreamResult(writer));
    }
```

We will run the Web application using the default XSL file as follows:

1. Right-click **LoginForm.html** and from pop-up menu select **Run on Server**.

> **Note:** In the Server selection dialog, make sure that you select **WebSphere Version 5.0 Test Environment**

2. Run the **`LoginForm.html`** file with the following values (Figure 6-9).

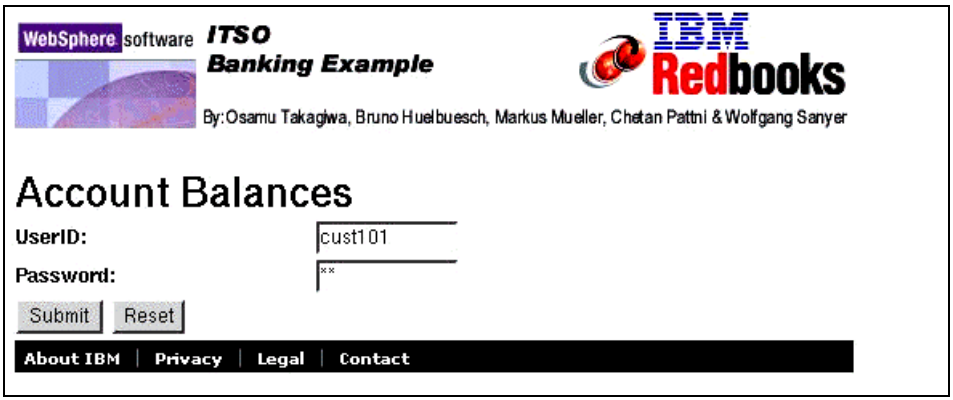


Figure 6-9 Login form page

3. The HTML output (Figure 6-10) was created by translating XML data and returning the customerBalance.xsl file while using SQLtoXML class. This is the default HTML page with the integrating the ITSO Banking example components.

FIRSTNAME	LASTNAME	BALANCE	ACCTYPE
Wolfgang	Sanyer	10000.00	CHECKING
Wolfgang	Sanyer	98726.26	SAVINGS
Wolfgang	Sanyer	5726.26	SAVINGS

Figure 6-10 Result page account balances (default)

We will show you how to run the Web application using the ITSO Banking example XSL:

1. Modify the customerBalanceNew.xsl in the Web project. This represents the HTML output. It is a good idea to compare with the older customerBalance.xsl file.
2. Change the servlet name in the XSL file.
3. Open the **CustomerBalanceReportsServlet** and rename the string `customerBalance.xsl` to `customerBalanceNew.xsl` in the getXSL() method:

   ```
   URL url = getURL(req, "customerBalanceNew.xsl");
   ```

4. Save the servlet and restart the server test environment.

> **Note:** Changes will not be activated until the project is restarted. It is preferable to stop the server test environment, then start it again.

5. Once the server is running, run the **`LoginForm.html`** file with the following values (Figure 6-11).

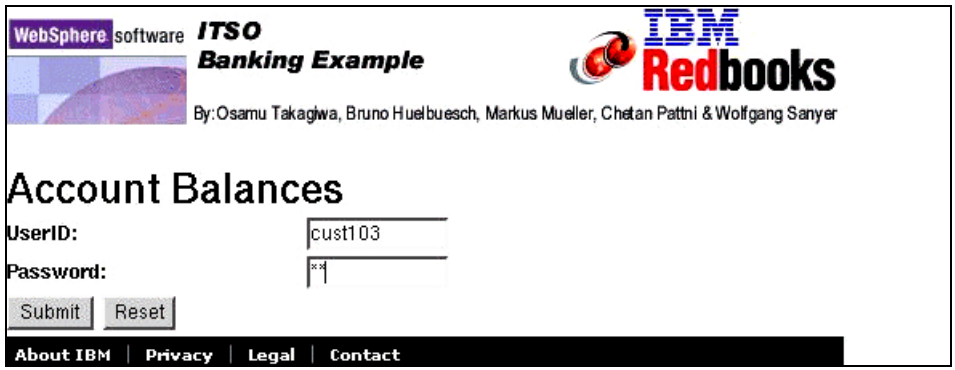


Figure 6-11 Login form page

Figure 6-12 is the HTML output.

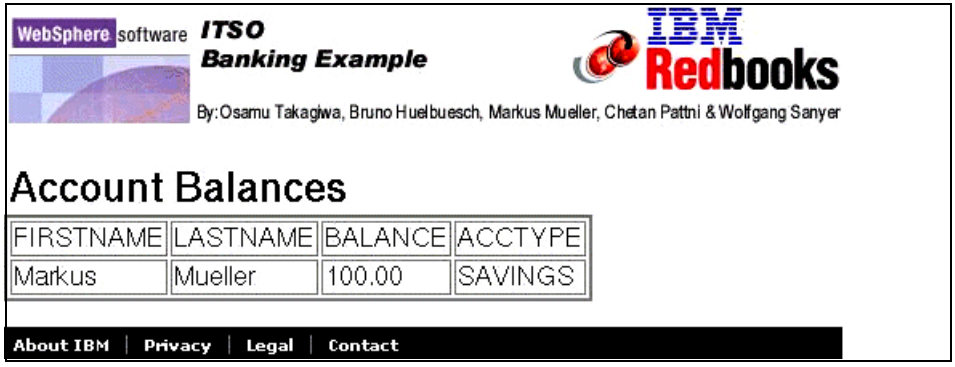


Figure 6-12 Result page of account balances (ITSO Banking example style)

6.5 Using the XSL debugger and transformation tools

The XSL debugging and transformation tool helps you debug or transform XSL stylesheets. Before you can use the debugger or transformation tools, you have to apply the XSL stylesheet to a source XML file, and generate some HTML output. You can open the HTML file in a Web browser from the Session view in the XSL Debug perspective.

The XSL debugging transformation tool records the transformation generated by the Xalan processor. This processor transforms XML files into HTML, text, or

other XML file types. It implements the W3C Recommendations for XSL Transformations (XSLT) and the XML Path Language (XPath).

To use the debugger follow these steps:

1. Select the **customerBalanceNew.xsl** and **customerBalance.xml** files from the Web project (Figure 6-13).

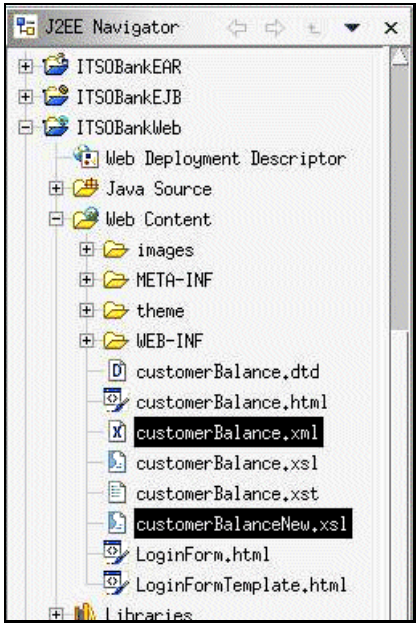


Figure 6-13 XSL and XML file for the transformation into HTML

2. Right-click on the **selected files**; from the pop-up menu select **Apply XSL as HTML**. This action will open the XSL Debug perspective (Figure 6-14). A new output HTML file is generated and located in the Web content directory under the name `customerBalance_customerBalanceNew_transform.html`.

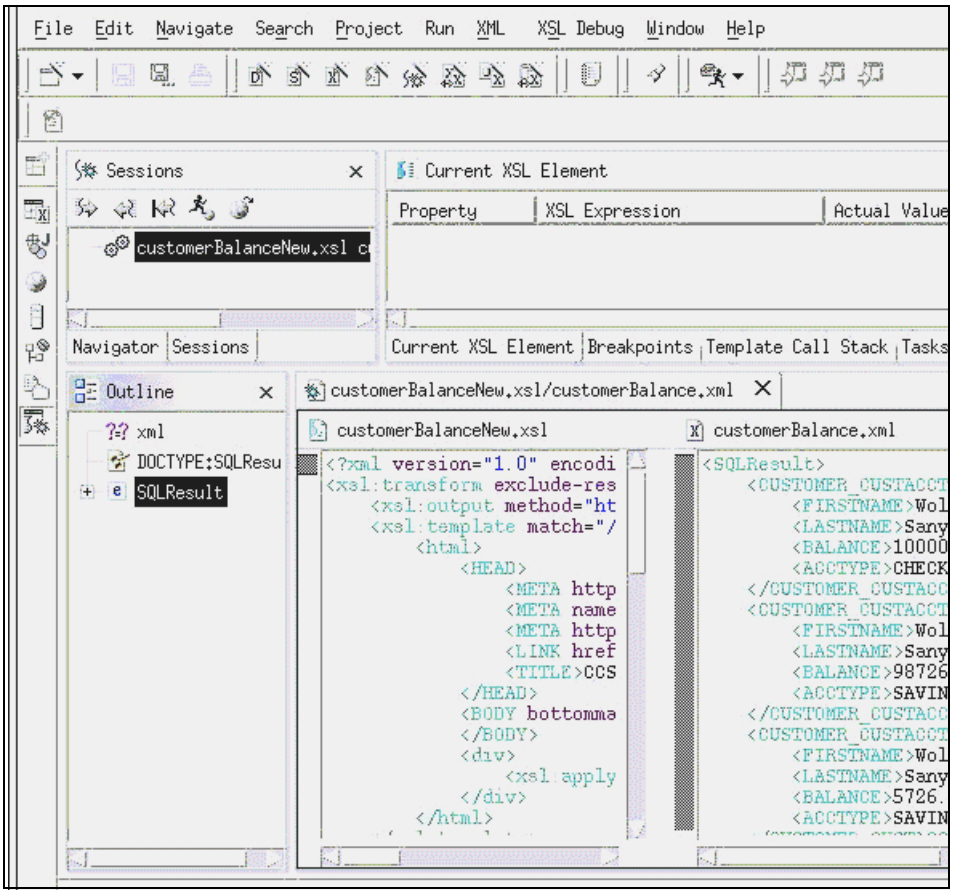


Figure 6-14 XSL Debug perspective

– The **Session view** contains a list of XSL debug sessions. The functions of the navigator buttons are:
 - Step forward 
 - Step backward 
 - Restart trace 
 - Run to breakpoint 
 - Open Web browser 

– **Current XSL Element view**
 - This view is your monitor of the transformation. All the informations about the XSL elements are shown here, line numbers of both XML and XSL, also attribute values at the current line number.

- **Source view of XSL/XML**
 - This view shows a currently transformed executed XSL element and XML element. In the XSL file source you can set several breakpoints to walk through the transformation.
- **Outline view**
 - This view is a upper navigation view of the XSL file, it allows easy navigation over the XSL source.

3. You can set known breakpoints in the XSL Source view and step over the XSL code elements to show the association with the XML file.
4. Finally, click **Open the Web browser** in the Session view to see the HTML file resulting from this XSL transformation.

6.6 Motivation to use XML/XSL instead of JSP

This section will show you the advantages of using XSLT technology instead of JavaServer Page technology in the Web development with Struts. Struts is an innovative server-side Java Framework designed to build Web applications, and is hosted by the Apache Software Foundation's Jakarta Project. This section shows the drawback of JavaServer Pages in Struts, and gives advice to use an open source Framework called StrutsCX for replacing JSP (JavaServer Pages) with XML and XSLT to better separate logic and presentation.

6.6.1 Struts with JavaServer Page drawbacks

The following drawbacks are represented to use Struts with the JavaServer Page technology:

- The developer can be embedding application logic into the JSPs.
- The JSP syntax is not XML compliant to guarantee that the resulting XML or HTML output will be well formed.
- Developers have to learn new APIs (Struts tag libraries) and it takes time.
- Recompile time after every change, which can be time consuming with JSPs.

The solutions for these problems can be:

- Separate logic and presentation.
- Enforce well-formed XML and HTML.
- Ease the separation of different view aspects, like layout and style.
- Allow for a faster development cycle.

These solutions can be realized when using Struts with the XSLT technology called StrutsCX. This is Struts with Castor XML and XSLT without any JavaServer Pages. It represents a Framework of classes to replace the JSP technology. StrutsCX is also open source and can be downloaded from the following Web site:

http://it.cappuccinonet.com/strutscx/index.html

7

Building a Web application with Ant

This chapter provides an overview of a powerful tool called Ant. Ant is used to build and deploy any Java application with or without an integrated development environment for Linux. This covers the following sections:

- Philosophy of Ant, which describes the philosophy and advantages of using Ant in a homogenous environments
- Setting up your environment to use Ant shows how to install and set up Ant on Linux. We will also show how to use Ant.
- Building a J2EE application with Ant shows how to build the ITSO Banking example using Ant in a stand-alone environment. We will be using the Web and EJB project from Chapter 5, “Enterprise JavaBeans 2.0” on page 59.

© Copyright IBM Corp. 2003. All rights reserved.

7.1 Philosophy of Ant

Ant is a build tool that enables you to automate the build process. In this case, Ant is similar to the make tool *make*. Ant was designed specifically for Java development and to support the deployment process. It is an open source, platform, independent, and it is a Java build tool. It uses XML for data manipulation and Java as the execution language. Since it uses Java, Ant is inherently extensible. It can be used in small to large projects, or any software project that requires integrating many components.

The following lists the reasons why we think Ant is a great build tool.

- It has a very simple syntax that is easy to learn. For XML users, Ant is easy to use.
- It is cross-platform because it can be used in any environment
- It has build support for J2EE development, for example, EJB compilation and packaging.
- It supports the deployment process outside the development environment. Ant gives the ability to automate the deployment process to other application servers via FTP and Telnet.
- Open source Java project such as Apache, Tomcat, or Cactus use Ant as their standard build environment.
- Ant supports Java-IDEs like NetBeans, Eclipse, jEdit, and WebSphere Studio Application Developer.

Ant was developed by the Apache Software Foundation as part of the Jakarta Project. Ant Version 1.5.1 is now released and can be downloaded from the following Web site:

`http://jakarta.apache.org/ant`

Note: Ant Version 1.4.1 comes with WebSphere Studio Application Developer 5.0. You can update Ant in WebSphere Studio Application Developer.

7.1.1 Build process approaches

In an ideal world, the environment in which you develop would be homogenous. For example, same operating system, methodology, and tools. Unfortunately, this is not the case in most projects. Therefore, we have different ways that we can approach a build process to develop an application.

Stand-alone

In a stand-alone procedure, developers deliver source code, along with the respective build files for their components. This code would most likely be delivered to a versioning system like CVS or ClearCase. A systems administrator would check out all the code and build it using a master build file. Many development teams use Ant under Linux to develop their components without a Integrated Development Environment (IDEs). This is a useful approach to have a build/deployment process to support the whole development.

Integrated in WebSphere Studio

WebSphere Studio Application Developer integrates all the necessary tools to effectively develop software at every stage in the software development cycle. The tool provides the Ant support as a built-in feature. Some functions are not supported (ejbDeploy, earExport, warExport) as Ant tasks generally are in WebSphere Studio. It is possible to install an ANT extra plug-in to have those functionalities.

Headless

In some situations, it might be necessary to run the build process outside WebSphere Studio, but still keep its dependencies. In these situations, a Java wrapper can be built around Ant to understand these dependencies. One is provided at:

http://www7b.boulder.ibm.com/wsdd/library/techarticles/0203_searle/searle.html

7.2 Setting up your environment to use Ant

The latest stable version of Ant is available at the Ant Web page:

http://jakarta.apache.org/ant

Ant can be used successfully on several operating systems like Linux, Solaris, Windows, MacOS X etc. To work with Ant you also need a JDK installed on your system, at least Version 1.1 or higher. In our installation we used the JDK of IBM Version 1.3.1.

Here are the steps:

1. Get the newest version from the Apache Ant homepage. We used the actual version number 1.5.1. Download the binary GNU zip file in your download directory. (It is also possible to download and install the rpm image file.)
2. Uncompress the GNU zip file and untar the tar file to the /usr/local directory:

```
# gzip -d jakarta-ant-1.5.1.tar.gz
```

```
# tar -xvf jakarta-ant-1.5.1.tar
```

The build tool Ant is now installed on the */usr/local* directory. To work with Ant you have to set up your environment variable on Linux. Each user has his own file to set the environment. This file calls .bashrc_profile and is executed for every instance of the bash shell. Set up your .bashrc_profile file with the following parameters. (See Example 7-1.)

Example 7-1 Set up the bash_profile file

```
ANT_HOME=/usr/local/jakarta-ant-1.5.1
JAVA_HOME=/opt/IBMJava2-131
PATH=$PATH:$JAVA_HOME/bin:$ANT_HOME/bin
CLASSPATH=$CLASSPATH:$ANT_HOME/lib/ant.jar:$ANT_HOME/lib/optional.jar:$ANT_HOME
/lib/xml-apis.jar:$ANT_HOME/lib/xercesImpl.jar
export ANT_HOME, JAVA_HOME, PATH. CLASSPATH
```

Call your user profile to setting up your environment, after run the Ant command to get the help messages back:

```
$ . .bashrc_profile
$ ant -help
```

7.2.1 Basics of using Ant

The Ant build scripts are build files and are written in XML. Every build file (build.xml) contains one *project* element. A project element contains *target* elements, and each target consists a set of *task* elements.

The task elements performs functions such as moving a file; compiling a project; creating a JAR package; or creating an EAR package.

A property has a name and a value. It is referenced by ${<property.name>} and can be used in the whole build file. Property values can be set inside a build file or can be obtained externally from a property file. The advantage of property files is that you have different properties for only one build file. You only change the property file for changing your environment. You do not need to modify the build file. All operating system paths should be inside this file. This is useful for different test and development environments.

A path is a set of directories or files. The Java compilation task uses a path reference to determine the class path to use.

In our ITSO Banking example, we want to use the following build-in tasks (Table 7-1).

Table 7-1 Build-In tasks of ITSO Banking example

Build-in task	Description
ant	Runs Ant on a supplied build file
move	Moves a file/directory to a new file/directory. The destination file if overwritten if it already exists by default adjustment
echo	Output a message to the current System.out
delete	Deletes files or directories or a set of files specified by one or more file sets
javac	Compiles a Java source tree, the source and destination directory will be recursively scanned for Java source file to compile
mkdir	Creates a directory
tstamp	Sets properties containing date and time information.
cvs	Handles packages/modules retrieved form a CVS repository
jar	Jars a set of files in a Java archive file
war	Creates a Web application archive including files that should end up in the WEB-INF/lib, WEB-INF/classes or WEB-INF directories
ear	Creates an enterprise application archive file similar to the JAR task. This file includes the whole enterprise application.
exec	Executes a system command with Ant under the specified operation system

Run Ant

After the Ant installation, we can execute a build file (*build.xml*) by simply typing **`ant`** at the command line prompt:

```
$ ant [options] [target1] [target2] [target3] ...
```

If a build file is not named *build.xml*, you will need to specify the build file name by using the **`-buildfile`** option:

```
$ ant -buildfile [xml file] [target]
```

Otherwise, you can run it only with:

```
$ ant
```

Ant invokes the help function to see additional command line options:

```
$ ant -help
```

7.3 Building J2EE applications with Ant

This section gives a general overview on how to build a J2EE application with the build tool Ant shown in the ITSO Bank example. These will mainly meet the requirements of developers and deployers who are building large scale J2EE applications with source code delivered by different teams or modules. For this community, it is interesting to have a powerful and simple build tool to reproduce the whole application. Ant is created to support the development aspect, but it also can be used in bigger projects as a build and deployment tool.

The ITSO Bank example is a J2EE Enterprise Application. This application includes a Web project and a EJB project, and is stored in CVS (Concurrent Versions System) as module names (Table 7-2). We use the CVS system to get all source code and configurations files out to start the ITSO Banking build file on a Linux environment.

Table 7-2 Mapping J2EE project to CVS module

Project name	**CVS module name**
Enterprise Application Project	ITSOBankEAR
Web Project	ITSOBankWeb
EJB Project	ITSOBankEJB

The structure of the CVS module name also describes the directory structure on the Linux environment like the J2EE naming conventions. On the Linux file system, we will be creating the following structure for the build process (Figure 7-1).

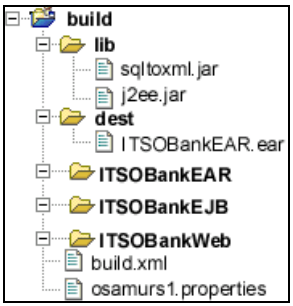


Figure 7-1 Flie system structure of the build directory

The build file is named *build.xml* and uses the file *osamurs1.properties* to provide environment information in Linux. The property file contains all important class paths to build the J2EE application. In order to modify or add the class paths to the build process, it must be done in this property file. The build file *build.xml* has the following targets inside as shown in the Table 7-3.

Table 7-3 Target names of build.xml

Target names	Description
init	Only the message to start the build with the actual timestamp
clean	Remove all generated files - used to force a full build
getcvs	Export of all needed project with the tag version itso_3 from the cvs repository, the modules are ITSOBankEAR, ITSOBankWeb and ITSOBankEJB
web-compile	Compiles Java sources form the ITSOBankWeb project into .class files
ejb-compile	Compiles Java sources form the ITSOBankEJB project into .class file without deployed Java code
ejb-jar	Creates the deliverables JAR file of the EJB project, it depends on the ejb-compile
ejb-deploy	Generates deployed code for the WebSphere Application Server, it calls the deploy tool “ejbdeploy”
war	Creates the deliverable war file of the Web project
ear	Creates the ear of the whole project within all other deliverable archive files

The next subsections describes more detail of the targets in the build file.

Setting properties and path elements

The most important concept to understand working with Ant is the meaning of the properties. Properties are loosely analogous to variables in that they are mapped between names and values, and are very similar conceptually to java.util.Properties. Ant properties depend on the context of their use, denoted by ${*property.name*} within the build file.

The property task allows a build file to define their own sets of properties. The variants that we use are the name/value attribute, and loading a set of properties from a properties file.

We used a property tag in our build file to load the configuration and setting data into our build process. The property file osamurs1.properties contains all needed settings of the build environment to build the J2EE application without any written paths in the build file. To load the property file, we use the property task. (See Example 7-2.)

Example 7-2 Loading the property file osamurs1.properties

```
<project name="ITSOBankingExample" default="ear">
<property file="osamurs1.properties" />
...
```

The values in the properties file may also contain property references. We consider some lines in our property file osamurs1.properties. (See Example 7-3.)

Example 7-3 Some code lines of the properties file

```
...
# Properties of the ITSOBankWeb project
web=ITSOBankWeb
web.project=${home.build}/ITSOBankWeb
web.sources=${web.project}/Java Source
web.content=${web.project}/Web Content
web.classes=${web.content}/WEB-INF/classes
...
```

If the file is loaded in the build file, you can refer to all properties in this file to make the build file independent to any environment or operating system. This is also helpful for the development in different teams. The teams can create their own build file.

Note: In the example we used all environment variables from the properties file as an input for the build file. The build file is independent and can be used in other applications.

To set external libraries we need to have the directory called *lib*. In this directory there are all the external libraries that we need in our application. These libraries have to be available at the build time (runtime). We need to use one library (j2ee.jar) to compile the source code of our project.

The classpath can be simplified in the build file for the project. We use the path element tag, which contains the pathement tags to set the location of the common libraries to compile on runtime the Java source code. In our example we stored the common libraries in the lib directory if they are not set in the class path of the user (Example 7-4).

Example 7-4 Common library path

```
...
<path id="common.classpath">
   <pathelement location="${lib.j2ee}" />
</path>
...
```

Target init

The first target to execute is the init target (Example 7-5). All other targets in our build file depend upon on this. It contains the *<tstamp>* task to set up properties, which include the timestamp information like date and time. The properties are available for the whole build file. To write timestamp information as a message on the screen, we used the *<echo>* tag to show the time and the starting date of our build file.

Example 7-5 Init target in the build file

```
<target name="init">
<tstamp/>
<echo message="Build of the J2EE Project Banking example started at ${TSTAMP}
on ${TODAY}" />
</target>
```

To start only the target init on the command line type:

```
$ ant init
```

The result of the command will be the current time and date. (See Example 7-6.)

Example 7-6 Output after running the init target on command line

```
Buildfile: build.xml

init:
```

```
     [echo] Build of the J2EE Project Banking example started at 1155 on
October
 19 2002

BUILD SUCCESSFUL
```

Target clean

The next target after the initialization is the clean target. In our example we clean up our directory structure. This is very important to start a new build without any older dependencies from another build cycle before. The *<delete>* task deletes the modules and the *dest* directory. These directories contain all needed packages and modules from a build cycle before. After this cleaning, we are ready to start a new build.

Example 7-7 Clean up environment for the next build cycle

```
<target name="clean" depends="init">
   <echo message="Clean all" />
   <delete dir="${ear.project}" />
   <delete dir="${web.project}" />
   <delete dir="${ejb.project}" />
   <delete dir="${dest}" />
</target>
```

Target getcvs

During the development of this book, we used a CVS server as our repository for coding the samples in WebSphere Studio. The J2EE project structure is the same as the module structure in CVS. We have to update our build structure under Linux with a new code level to build the J2EE application. Before we can do this, you have to create a code base line in CVS called *tag version*. This version is now our driver to build the ITSO Bank example. In our example we set the properties in the property file to configure the connection to the CVS server.

Example 7-8 CVS properties

```
# CVS Properties
cvs.home=:pserver:osamurs1:osamurs1@dhcp39039.almaden.ibm.com:/home/cvs/cvsrep
cvs.command=export -r itso_1
ear=ITSOBankEAR
war=ITSOBankWeb
ejb=ITSOBankEJB
```

Note: The J2EE directory structure below the module directories correspond to the WebSphere Studio.

Ant supports the *<cvs>* task for working with the CVS server, and all operations to get out the source code from the repository:

- **cvsRoot**:Root directory of the remote CVS server
- **package**: Name of the CVS module to extract (also checkout)
- **dest**: Directory other than the project's root directory, where files are extracted locally

Example 7-9 getcvs target in the build file

```
<target name="getcvs" depends="clean">
   <echo message="Get file from CVS Repository: ${cvs.home}" />
   <cvs cvsroot="${cvs.home}" package="${ear}" dest="${home.build}"
command="${cvs.command}" />
   <cvs cvsroot="${cvs.home}" package="${web}" dest="${home.build}"
command="${cvs.command}" />
   <cvs cvsroot="${cvs.home}" package="${ejb}" dest="${home.build}"
command="${cvs.command}" />
   <mkdir dir="${dest}" />
</target>
```

Target Web-compile of ITSOBankWeb

Before we compile the code of the Web application, we first ensure that the destination directory exists. We achieve this using the *<mkdir>* task to create the *classes* directory of the Web application. The *<copy>* task copies all need files excluding Java sources from the Java source directory into the classes directory. This function is needed if some configuration files are used in the Java source directory. After we start with the *<javac>* task the compile of the Java code in the Web application directory, specifying with following attributes:

- **srcdir:** The location of the Java source files
- **destdir:** The directory where to writer generated .class files
- **classpathref:** A reference to the class path to use during compilation
- **optimize:** Use a compiler optimization
- **debug:** Generate debug information in all class files
- **deprecation:** Output source locations where deprecated APIs are used

The compile of the Web-compile task is shown in Example 7-10.

Example 7-10 Target web-compile

```
<target name="web-compile" depends="getcvs">
   <mkdir dir="${web.classes}" />
   <copy todir="${web.classes}">
      <fileset dir="${web.sources}" excludes="**/*.java"/>
   </copy>
   <javac srcdir="${web.sources}"
          destdir="${web.classes}"
          classpathref="common.classpath"
          optimze="${javac.optimize}"
          debug="${javac.debug}"
          deprecation="${javac.deprecation}"
</target>
```

Note: An Ant itself is not a Java compiler; it simply contains a facade over compilers such as IBM's Javac.

Target ejb-compile of ITSOBankEJB

This compile target for the EJB is nearly identical to that for the Web application code. The difference is the destination directory for the compiled .class files. All Java source files and compiled Java classes are located in the same directory called *ejbModule*.

Example 7-11 Target ejb-compile

```
<target name="ejb-jar" depends="ejb-compile">
      <echo message="Create JAR file for EJB" />
      <jar destfile="${ejb.jar}" manifest="${ejb.meta}/MANIFEST.MF">
         <fileset dir="${ejb.module}">
         </fileset>
      </jar>

</target>
```

Target ejb-jar of ITSOBankEJB

To create a JAR file of the EJB project, we use the Ant *<jar>* task to package the EJB JAR file. The *<fileset>* task specified the directory to package. We stored this jar file in the ITSOBankEJB directory to generate later deployed Java code. Now we have only undeployed Java code in the JAR file.

Example 7-12 Target ejb-jar

```
<target name="ejb-jar" depends="ejb-compile">
   <echo message="Create JAR file for EJB" />
   <jar destfile="${ejb.jar}">
```

```
      <fileset dir="${ejb.module}">
      </fileset>
<  /jar>
</target>
```

Target ejb-deploy of ITSOBankEJB

The previous section described how to generate an undeployed EJB JAR file. Before we can install the EJBs on an application server like WebSphere, we have to generate the deployed code of the EJB JAR file. The deployment process involves examining the EJB code and deployment information, and generating and compiling WebSphere-specific code that links the EJBs with the WebSphere EJB container implementation. This target runs the WebSphere tool *ejbdeploy* with the undeployed EJB JAR file as input, and generates a new JAR file containing the deployed code. To start the command line tool ejbdeploy, we have to be a root user or a special user with root permission in the Linux environment. The Ant *<exec>* task helps to call the ejbdeploy tool with the following arguments:

- Name of the input JAR file (undeployed code)
- Name of the working directory
- Name of the output JAR file (deployed code)
- Option **`-quiet`** used only to display errors

Example 7-13 Target ejb-deploy

```
<target name="ejb-deploy" depends="ejb-jar">
   <echo message="Create EJB deployed code for WebSphere using the ejbdeploy
tool" />
   <exec executable="${was.ejbdeploy}">
      <arg value="${ejb.jar}"/>
      <arg value="${ejb.project}"/>
      <arg value="${ejb.project}/ITSOBankEJB_deployed.jar"/>
      <arg value="-quiet"/>
   </exec>
   <move file="${ejb.project}/ITSOBankEJB_deployed.jar"
tofile="${ear.project}/ITSOBankEJB.jar" />
</target>
```

The Ant *<move>* task moves the deployed JAR file to the ITSOBankEAR directory. Later this directory will be package to the ITSOBankEAR .ear file. There is only one way to use Ant as a build tool to generate deployed code. It is also possible to run the command line for *ejbdeploy*, or to start the Application Assembly Tool within WebSphere Application Server.

Note: The *ejbdeploy* tool does not actually deploy EJBs into the application server, it means it does not install them into an instance of the Application Server.

The deployed code includes the RMI (*Remote Method Invocation*) stub code generated by the rmic compiler.

Target WAR of ITSOBankWeb

The Web application has to be package in a WAR file. A WAR file is a JAR file with an extended format; a WEB-INF folder contains classes and a lib folder containing libraries. The web.xml file in the WEB-INF directory describes the Web application on the application server; if this file is missing or invalid, the WAR file does not contain a Web application.

To generate the WAR file in Ant we use the *<war>* task; this task is a subclass of the *<jar>* task, and it also supports all of the parent tasks's attributes and elements. The parameters of the *<war>* task in our example are:

- warfile: The name and location of the created WAR file
- webxml: The web.xml file to be used as a deployment descriptor for WAR file
- manifest: Specify the manifest file to use

The <fileset> task specifies a collections of files to be package. All files and directories under the Web content directory will be inside. The WAR file will be created in the ITSOBankEAR directory.

Example 7-14 Generate war file ITSOBankWeb.war

```
<target name="war" depends="ejb-deploy">
       <war warfile="${web.war}" webxml="${webxml}"
            manifest="${web.meta}/MANIFEST.MF">
          <fileset dir="${web.content}" />
       </war>

</target>
```

Target EAR of ITSO Bank example

Finally, we have to create the whole Enterprise Application as an EAR file. The WAR file of the Web application, and the JAR file of the EJB project are available in the ITSOBankEAR directory. We created an EAR archive of this directory with the Ant *<ear>* task. This tasks includes the following parameters:

destfile: The name and location of the EAR file (destination)

appxml: File to incorporate as application.xml

Example 7-15 Target EAR

```
<target name="ear" depends="war">
   <ear destfile="${dest.ear}" appxml="${appxml}">
      <fileset dir="${ear.project}"/>
   </ear>
</target>
```

The EAR archive file is now available in the *dest* directory and ready to install into the application server as Enterprise Application.

Note: The whole build and property file of the ITSO Bank example with build log is available in the sample directory in the sample code under the BuildWithAnt directory.

8

Deploying the Web application

This chapter describes the deployment of the J2EE application in the following sections:

- Deploying an Enterprise Application describes step-by-step the manual deployment and installation of the ITSO Bank example on the Application Server Version 5 with the administration console.
- Setting up a remote server describes the important steps to create a remote server instance to publish a Web application to the remote server.
- Automatic deployment by tools shows the way to use the **wsadmin** command tool and the Ant build tool to install the ITSO Bank example on the Application Server Version 5.

© Copyright IBM Corp. 2003. All rights reserved.

8.1 Deploying an Enterprise Application manually

This section describes the deployment of the J2EE application ITSO Bank example in the WebSphere Application Server Version 5 on Linux. It contains the following subsections:

- EAR export from WebSphere Studio Application Developer
- Starting the WebSphere administration console
- Configuration of the WebSphere resources
- Installation of the ITSO Bank EAR file
- Testing the application

8.1.1 EAR export from WebSphere Studio Application Developer

You have to export the file from WebSphere Studio as an EAR file to install it on the Application Server. Complete the following steps:

1. File menu in WebSphere Studio Application Developer, select **Export ...**
2. From the Export wizard select the **EAR file** (Figure 8-1).

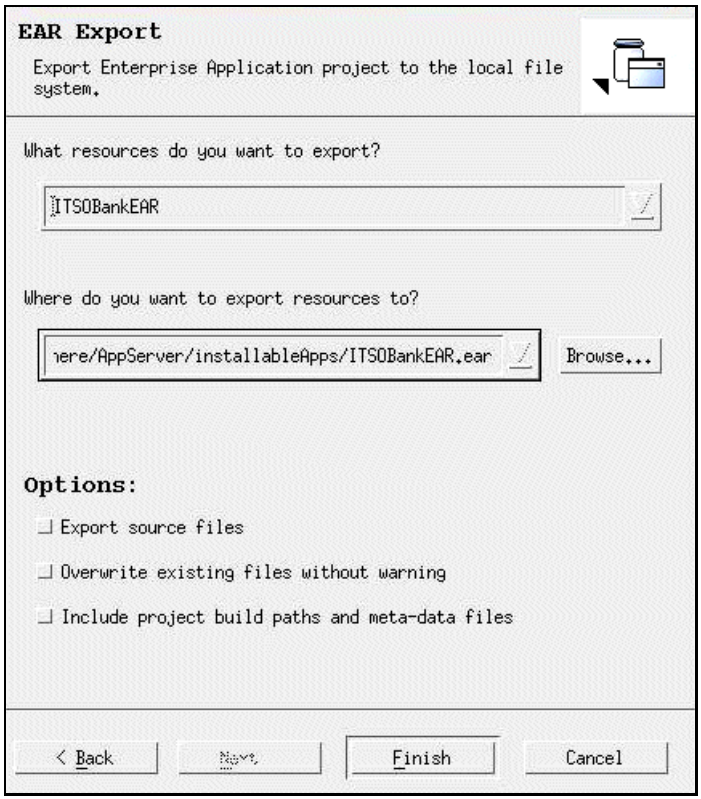


Figure 8-1 Exporting the EAR file

3. Select the **EAR Project** to export and define the location to export this file to the Linux directory (*/opt/WebSphere/AppServer/installableApps*). Choose **no additional options.**
4. Click **Finish** to export.

8.1.2 Starting the WebSphere administration console

To start the administration console on Linux, you have to be in root. Try to use the First Steps Tool tips to get easy access to the WebSphere Application Server. You have the opportunity to start or stop the application server, verify the installation, access to the InfoCenter, run the Application Assembly Tool, access the administrative console, access the Sample Gallery, or launch the product registration. In Linux you have to call the shell script *firststeps.sh* with:

```
#/opt/WebSphere/AppServer/bin/firststeps.sh &
```

The following window appears with the tool tips (Figure 8-2).



Figure 8-2 First Steps Tool Tips

Start the Application Server by selecting **Start the Server**; you have also the opportunity to start the server with the shell script **`startServer.sh`** and the argument *<servername>*. Instead of the tool, run the shell script on the command line. More detailed information is shown in 8.3 “Automatic deployment by tools” on page 170.

Once you have started the Application Server, start the administration console. Click **Tool Tip Administrative Console**; you also have the opportunity to access the console with the browser by typing:
`http://dhcp39065.almaden.ibm.com:9090/admin` in our example. Generally, take your own hostname. Figure 8-3 should appear.

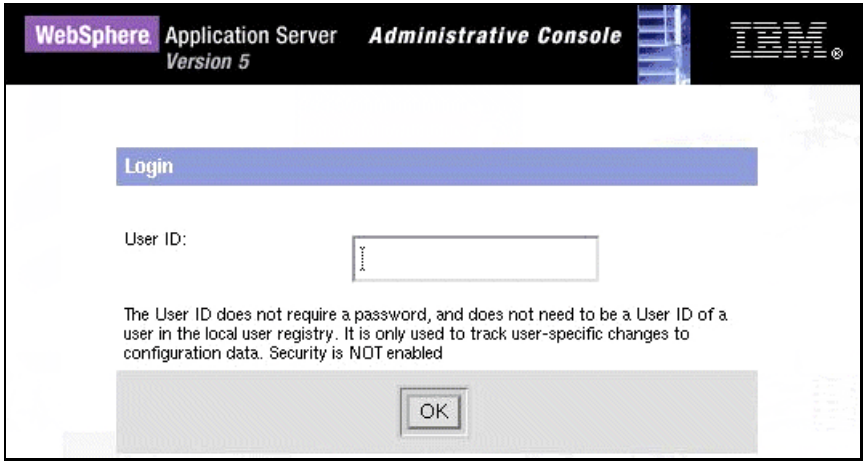


Figure 8-3 Administration Console login

The User ID does not required a password. This User ID is stored in a local user registry to track only the user-specific changes. Type one User ID (root) and click **OK**.

Now the Administration Console will be come up (Figure 8-4).

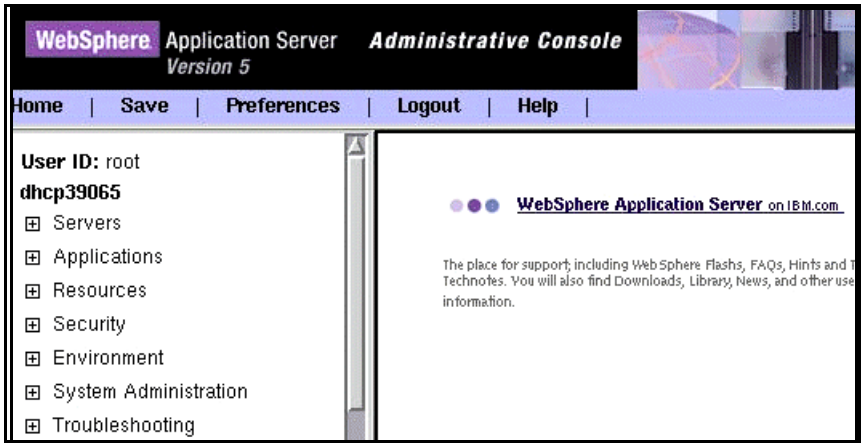


Figure 8-4 WebSphere Administration Console Version 5

8.1.3 Configuration WebSphere resources

To access the database you must first create a JDBC Provider. Use the administrative console to do this. You need to know where the database software is already installed, and where the JDBC drivers are located (Figure 8-5).



Figure 8-5 JDBC Provider

JDBC Provider: Perform the following steps:

1. Click **Resources** -> **JDBC Providers** link.
2. From the JDBC Providers frame, make sure the scope is set to `Node`.
3. Click the **New** button to create a new JDBC provider.
4. From the New JDBC Provider frame, use the drop-down list box to choose a JDBC Provider of DB2 JDBC Provider (XA).
5. Click the **Apply** button to view the settings page of your JDBC Provider. Enter the following properties:

 Name: `JDBC Provider (XA)`

 Native Library Path: `<db2_installation_directory>/lib i.e. /home/db2inst1/sqllib/lib`
6. Click the **Apply** button to update the changes.

Data source: The application uses a data source to access the data from the database. The data source is associated with the JDBC Provider that supplies the specific JDBC driver implementation class.

Do the next steps to create the jdbc/WSAD5 data source:

Note: Make sure that you want to use a Version 5 Data Source. If you are using the EJB specification 1.0 and servlet specification 2.2, you must use a Version 4.0 Data Source.

1. Form the JDBC Provider frame, scroll down and click the **Data Sources** link (Figure 8-6).

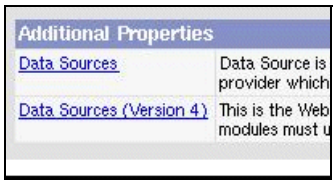


Figure 8-6 Additional Properties: data sources

2. From the Data Source frame click the **New** button.
3. From the Data Source Configuration frame enter the following properties:

 Name: `WSAD5`

 JNDI Name: `jdbc/WSAD5`

 Click **Use the Data Source in container managed persistence (CMP).**
4. Scroll down and click the **Apply** button.
5. Scroll down and click the **Custom Properties** link (Figure 8-7).



Figure 8-7 Custom Properties

6. From the Custom Properties frame click **databaseName.**
7. From the databaseName frame, change the Value field from `SAMPLE` to `WSAD5` for our databaseName (Figure 8-8).

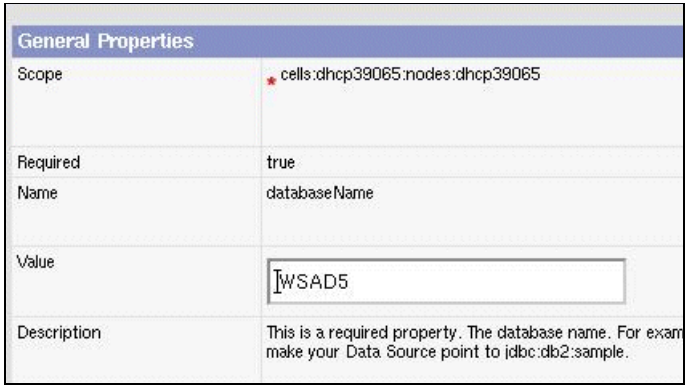


Figure 8-8 Change the database name value

8. Click **OK** button.
9. Next you have to specify a list of userids and passwords for use by Java 2 Connector security. You have to create a J2C Authentication Date Entries. Click **J2C Authentication Data Entries** in the Data Source frame (Figure 8-9).



Figure 8-9 J2C Authentication Data Entries

10. Click **New** in the **J2C Authentication Data Entries** page.
11. Fill in the fields on the resulting page like the Figure 8-10: The user ID and password form your database access.



Figure 8-10 Create a new authentication alias calls itsops

12. Click **Apply**, return now to the Date Source frame.
13. Set the new alias from the drop-down menu for Component or Container-managed Authentication Alias field (Figure 8-11), and click **Apply**. These aliases are used for the database authentication during runtime.

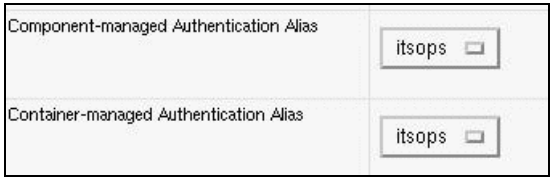


Figure 8-11 Set the authentication aliases in the Data Source frame

14. Click **OK**.

To modify DB2 Websphere variable do the following:

1. From the left-hand pane of the Administration Console, expand **Environment** and click **Manage WebSphere Variables** (Figure 8-12).



Figure 8-12 Environment: WebSphere variables

2. Form the WebSphere Variables frame, scroll down and click **db2_jdbc_driver_path** to set the value of your own environment.
3. Make sure that the DB2_JDBC_DRIVER_PATH variable has a value which points to your DB2 Java installation directory *<db2_installation_directory>/java*. This variable needs the JDBC Provider to find the actual JDBC driver in the configuration (Figure 8-13).



Figure 8-13 DB2 variable

4. Click **OK**.

All needed JMS configurations are described in Chapter 5. “Enterprise JavaBeans 2.0” on page 59, under the section “Set up Listener Port, Queue Names, and JNDI Mapping” on page 63.

At the end, you have to *save* the configurations of what you have done. Otherwise, all settings will be not active in the WebSphere configurations.

8.1.4 Installation of the ITSO Bank EAR file

This section discusses how to install the ITSO Bank EAR file in the Application Server using the administration console. During this task, you will install the whole J2EE application (like the .ear, .war, and JAR files)

To install the application take the following steps:

1. Start the WebSphere Administration Console Version 5.
2. From the left-hand frame, expand **Applications**. Click the **Install New Application** link (Figure 8-14).



Figure 8-14 Install new application

3. The `Preparing for application install` frame comes up (Figure 8-15). Use the **Browse ...** button to specify the directory where you stored the EAR file before (in our example: */opt/WebSphere/AppServer/installableApps*). Choose the **Local path.**

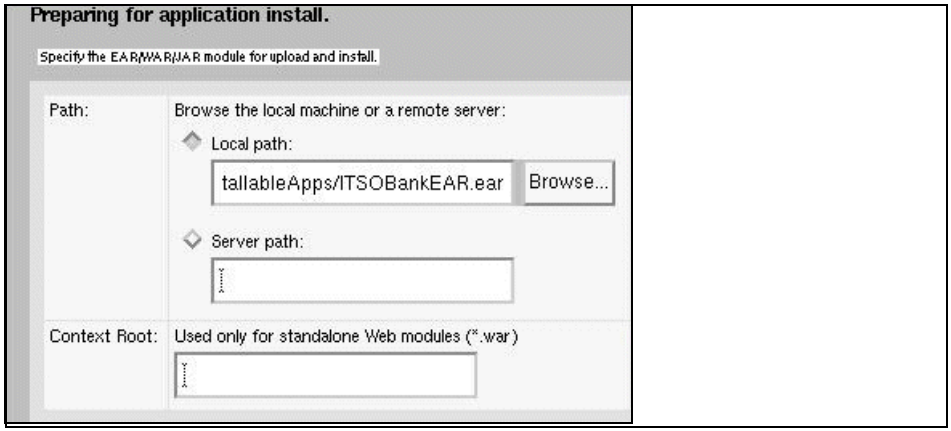


Figure 8-15 Preparing for application install

4. Click the **Next** button.
5. From the Default bindings frame, take the defaults setting and click **Next.**
6. Now walk through seven steps. From the *Step 1* in the Preparing for application frame, specify a directory to install the application. (In our example: */opt/WebSphere/AppServer/installedApps/dhcp39065*. The dhcp39065 directory is generally the nodename where the Application Server is installed. Type **`ITSOBankEAR`** in the field Application Name (Figure 8-16).

Note: The quantity of the steps depends on the contents of the EAR file. In our example we have seven steps. If you have only a Web application to deploy, you might be have less than seven steps.

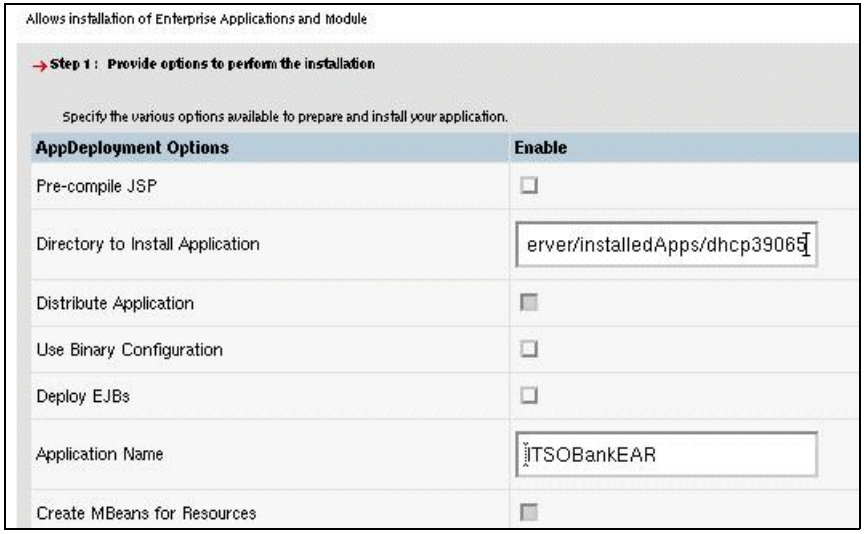


Figure 8-16 Step 1: Deployment options

7. Click **Next** to come to the `Step 2 Provide Listener Ports for Messaging Beans` frame make sure the Listener Port is set to `LP1`.

> **Note:** Remember that one port is mapped to one Message-driven Bean (defined in 5.1.1, “Java Message-driven Beans” on page 60.)

8. Click **Next** to come to the `Step 3 Provide JNDI Names for Beans` frame, accept the JNDI name settings like the setting below (Figure 8-17).



Figure 8-17 Step 3: Provide JNDI names for beans

9. Click **Next** to come to the `Step 4 Map virtual hosts for Web modules` frame, accept the default setting.
10. Click **Next** to come to the `Step 5 Map modules to application servers` frame; here you specify for each module a target server or a cluster. Accept the default setting.
11. Click **Next** to come to the `Step 6 Ensure all unprotected 2.0 methods have the correct level of protection` frame; the application contains EJB 2.0 CMP beans that do not have method permissions defined in the

deployment descriptor for some of the EJB methods. For methods marked unchecked, no authorization check is performed prior to their invocation. Accept the default setting.

12. Click **Next** to come to the `Step 7 Summary` frame; click **Finish** to install the application.

> **Note:** After clicking **Finish**, if you receive an Out Of Memory exception and the source application file does not install, your system might not have enough memory, or your application might have too many modules in it to install successfully onto the server.

13. You have to receive the message: `Application ITSOBankEAR installed successfully.`
14. Click **Save** on the administrative console taskbar to save the changes to your configuration.

> **Note:** If you do any changes in the administrative console, you must save the configuration, otherwise the changes will be lost.

15. Click **Logout** on the taskbar of the console. Change to the `First Steps` frame to `Stop the Server`, or use on the command line the shell script **`stopServer.sh`** in the bin directory of the WebSphere Application Server.

> **Note:** You do not need to restart the application server after each application installation. This step is only done when there are configuration changes.

8.1.5 Testing the application

You can now test your application on the WebSphere Application Server:

1. Start the Application Server again with the `First Steps` frame.
2. Open the Web browser with the specific URL:

 `http://localhost:9080/ITSOBankWeb/index.html`

8.2 Setting up a remote server

In this section we want to publish a Enterprise Application inside the WebStudio Application Developer to a remote server like WebSphere Application Server Version 5. With Application Developer you can deploy the application to a remote

server using a remote server instance and configuration. Before you start, you have to make sure that the remote server has the following software already installed:

- IBM WebSphere Application Server Advanced Edition 5.0 for multiplatforms
- IBM Agent Controller
- FTP server (optional)

8.2.1 IBM Agent Controller

This section describes the installation and configuration of the IBM Agent Controller on the remote server on Linux. We use the rpm-image to install this software on Linux to run the following command:

```
# rpm -ivh ibmrac-5.0.0.0-0.i386.rpm
```

The IBM Agent Controller is successfully installed under Linux in the directory: /opt/IBMRAC

Configure and running the IBM Agent Controller

Run the shell *SetConfig.sh* in the bin directory of the installation directory of the Agent Controller. Make sure you are a root user to start the process:

```
# /opt/IBMRAC/bin/SetConfig.sh
```

Set up the following locations to your environment:

- Installation directory of the Agent Controller
- JDK installation directory (JAVA_HOME)
- Installation directory of the WebSphere Application Server Version 5

After this configuration, you have to start the Agent Controller:

```
# /opt/IBMRAC/bin/RAStart.sh
```

The logging about the *service* is stored in the *servicelog.log* file in the config directory. If the RAServer processes are running, the Agent Controller is installed succesfully. You can check the process with the following command:

```
# ps -ef | grep RAServer
```

8.2.2 Creating a server for remote testing with Application Server

When you creating a server in the Server view of WebSphere Studio Application Developer, you can specify a remote server as server type if you want to wish to

test on a remote installation of WebSphere Application Server. In this case you have to do the following steps:

1. In the Server view of the Server perspective, create the server configuration for the new remote Application Server: **New —> Server Instance and Configuration**. Fill in and select in the fields like below (Figure 8-18):
 a. **Server Name:** Enter a display name for the new server.
 b. **Folder:** Enter a folder name for the server.
 c. Select **WebSphere Remote Server** as server type.

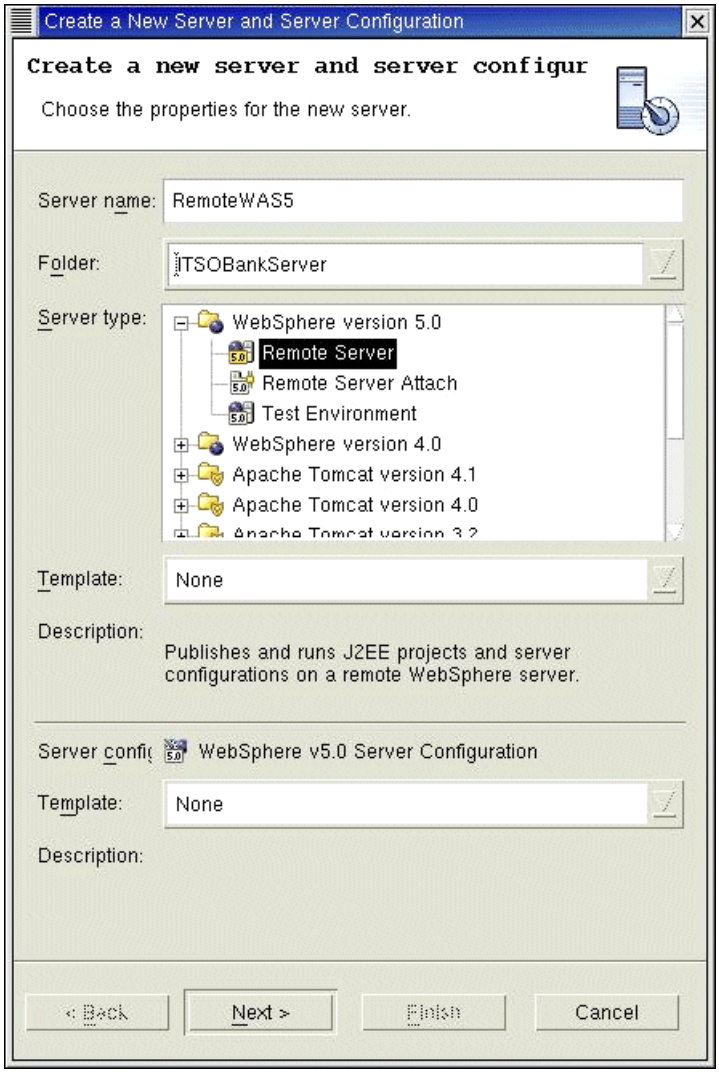


Figure 8-18 Creating a remote server instance

2. Click **Next**. The next window appears to specify the **Host address** of the remote server (Figure 8-19). Type the fully qualified DNS name, or the IP address of the remote server machine where WebSphere Application Server is running.

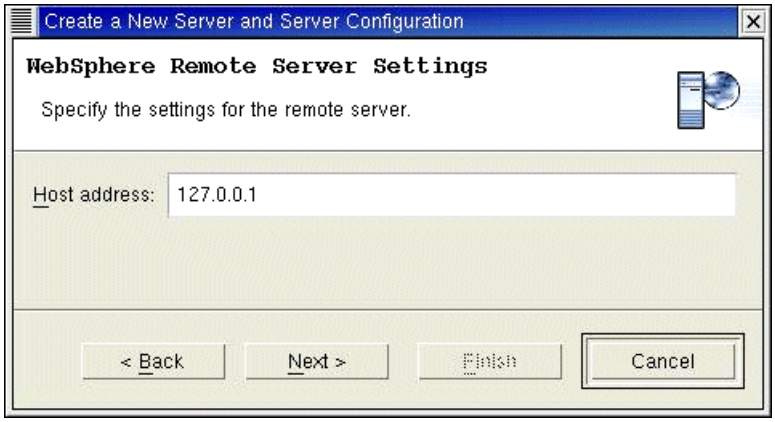


Figure 8-19 Setting the remote server host address

3. Click **Next**. In the next window (Figure 8-20), type in the WebSphere installation directory field in the path where you installed WebSphere Application Server on the remote machine. This path is the same like the $WAS_ROOT variable:

 a. **WebSphere Installation directory** is the directory where the Application Server is installed: i.e. */opt/WebSphere/AppServer*

 b. **Use default WebSphere deployment directory** check box when you creating a remote server to publish to this server. If do not want to use the WebSphere deployment directory, clear the check box and enter the WebSphere deployment directory field a new path directory name where the Web application and server configuration has to be published to the remote directory in the directories called config and installedApps.

 c. *Optional*: In the DB2 driver location field enter the DB2 location where the DB2 classes reside in the remote directory.

 d. Choose in the check box below the **platform of the remote machine**. It depends on which kind of platform your remote machine is running (for Linux choose **Other**).

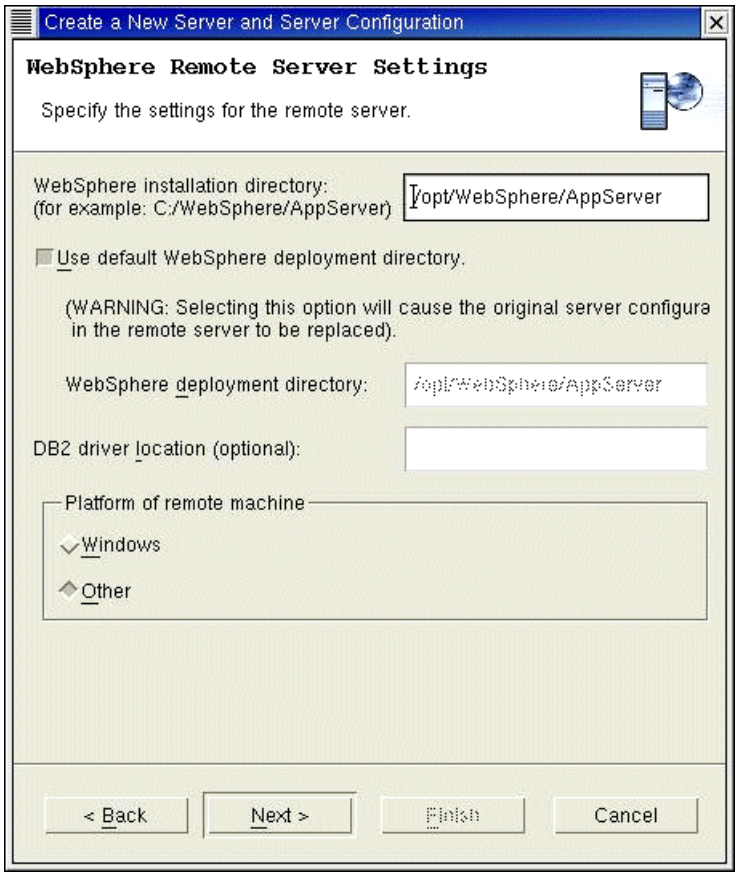


Figure 8-20 Remote server instance settings

4. Click **Next**. The following page allows you to create a remote file transfer instance (Figure 8-21).It contains information for transferring Web applications and server configurations to the remote server during publishing. Select one of the following radio buttons:

 a. **Create a new remote file transfer instance**, it defines a new set of parameters and environment settings needed to transfer files remotely.

 i. **Copy file transfer mechanism** to copy resources directly from one to another in the file system.

 ii. **FTP file transfer mechanism** copy resources from one machine to another using File Transfer Protocol (FTP)

 b. **Use an existing remote file transfer instance** lists the already defined remote file transfer instances that you use for transferring files remotely.

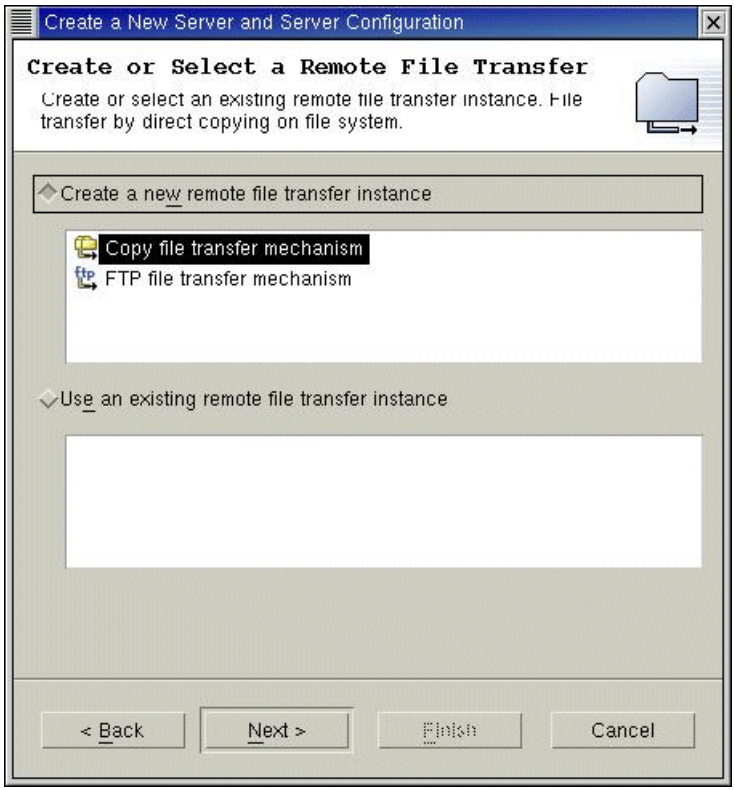


Figure 8-21 Remote file transfer option

c. If **Copy file transfer mechanism** is selected the next window appears with the following fields (Figure 8-22):

 i. **Project Folder**: Type the name of the project where the remote file transfer will be reside.

 ii. **Remote file transfer name**: Given from the wizard as remote file transfer name (possible to change).

 iii. **Remote target directory**: Type the remote target directory where you want to your applications and server configurations published. This remote target directory is the one seen by the local machine. If WebSphere Application Server installed on a different machine, then the remote target directory is the network drive that maps to the WebSphere deployment directory. If WebSphere Application Server installed on the same machine as the Workbench, then the remote target directory should be the same as the contents in the WebSphere deployment directory: i.e. */opt/WebSphere/AppServer*

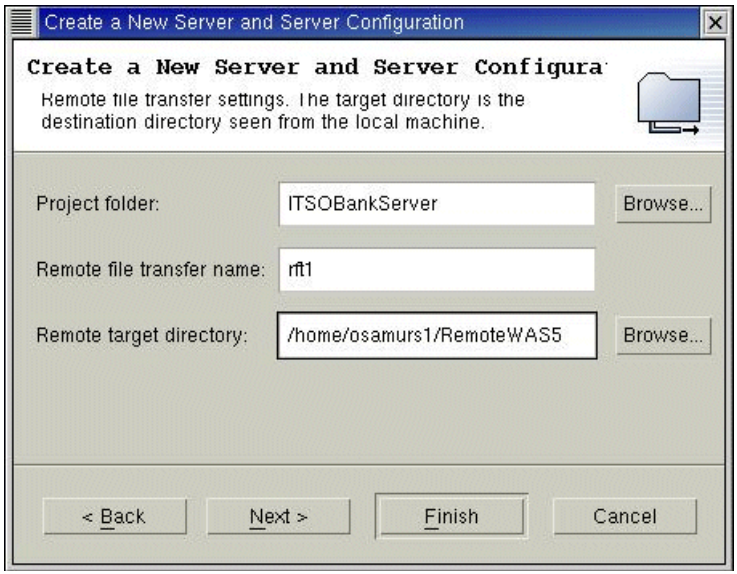


Figure 8-22 Remote copy options

iv. Click **ext** if you want to change the HTTP port number in the next window of the wizard.

v. Click **Finish** to create a remote server file instance and a remote server instance. The server instances appear in the Server view. The remote file transfer instance appears in the Navigator view of WebSphere Studio Application Developer.

d. If FTP file transfer mechanism is selected, the next window appears with the following fields (Figure 8-23):

i. **Project Folder**: Type the name of the project folder where the remote server will reside.

ii. **Remote file transfer name**: Given from the wizard as a remote file transfer name (Possible to change).

iii. **Remote file transfer directory**: Type the remote target directory where you want your application and server configuration published. This remote target directory points to the WebSphere deployment directory that is seen form the Workbench using the FTP client program.

iv. **FTP URL**: Type the URL that is used to access the FTP server.

v. **User login**: Type the FTP user ID used to access the FTP server.

vi. **User password**: Type the FTP password used to access the FTP server.

vii. **Connection timeout**: Type the time (in milliseconds) that the Workbench will wait attempting to contact the FTP server before timing out.

viii. **Use PASV Mode (Passive Mode) in go through the firewall**: Select this check box if you want pass through a firewall provided that one is installed between your FTP server and the Workbench.

ix. **Use Firewall**: Select the check box if you want to use the firewall options. To change the firewall options, click **Firewall Settings** to specify and settings.

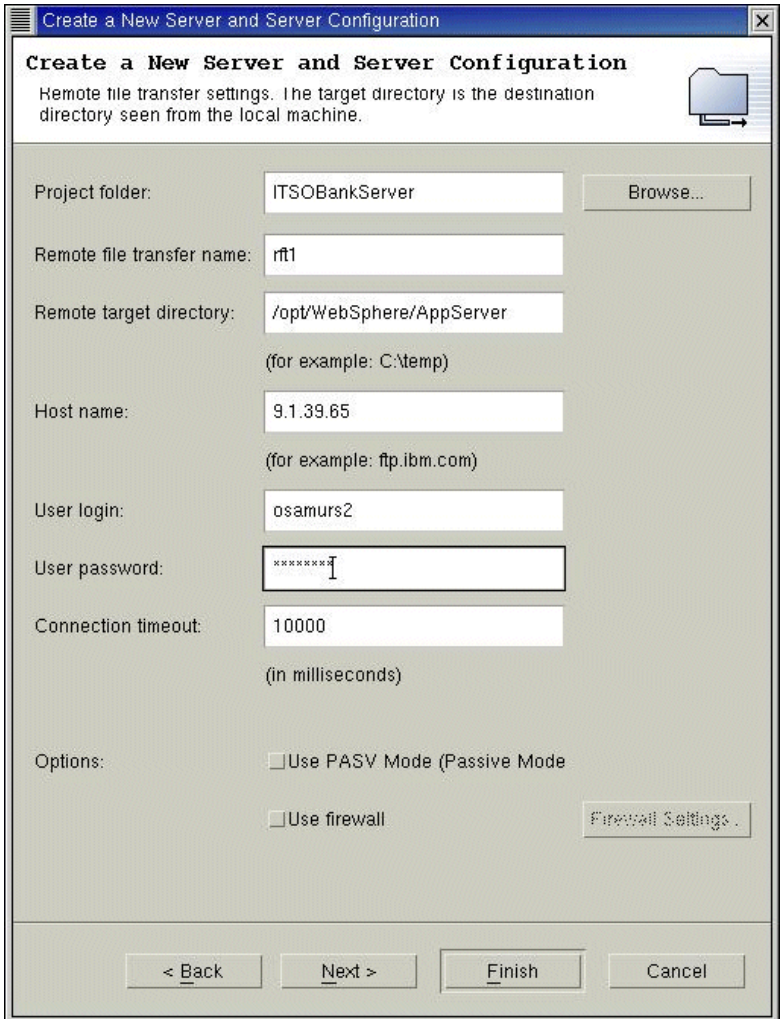


Figure 8-23 FTP configuration options

e. Click **Next** if you want to change the HTTP port number in the next window of the wizard.

f. Click **Finish** to create a remote server file instance and a remote server instance. The server instances appear in the Server view. The remote file transfer instance appears in the Navigator view of WebSphere Studio Application Developer.

8.3 Automatic deployment by tools

This section describes the deployment (installation EAR file) with the non-graphical WebSphere administrative (wsadmin) scripting program; the uses of the available command line tools for WebSphere Application Server; and shows the way to integrate the deployment process in the Ant script in Chapter 7, “Building a Web application with Ant” on page 135.

8.3.1 Installing application with the wsadmin tool

Scripting is a non-graphical alternative that you can use the configure and manage the WebSphere Application Server. One manual way is to use only the administrative console (Web GUI) to configure and install your Enterprise Application.

The Java application used for the WebSphere Application Server Advanced Edition Administrative Console in Version 4 has been replaced by a Web application, which editions use. The Version 5 Web-based console is much expanded and improved over the Web-based console used for the older version.

The WebSphere Control Program (WSCP) available in Version 4, has also been replaced by a more flexible and coherent facility (known as wsadmin) which is available in all Version 5 editions. You can use wsadmin in interactive or batch modes and perform any operations that you can do with the administrative console. The wsadmin utility is based on IBM’s Bean Scripting Framework (BSF) and supports a variety of scripting languages, including JavaScript, JPython, and Jacl (a Tcl derivative).

The wsadmin tool provides an ability to execute scripts. It supports a full range of product administrative activities. This tool is located in the *WebSphere/AppServer/bin* directory. To invoke a scripting or using the commands object, you have to call the shell script **`wsadmin.sh`**. We want to install the Enterprise Application ITSO Bank with the command object **`$AdminApp`**. We do not need to create a script to run this command.

Run the script commands as individual commands; invoke the **AdminApp object** command interactively to use the **wsadmin -c** command from an operating system command prompt.

The following command uses the EAR file and the command option information to install the ITSO Bank application:

```
# wsadmin.sh -c "\$AdminApp install <EAR file> {options}"
```

etc.

```
# wsadmin.sh -c "\$AdminApp install /build/dest/ITSOBankEAR.ear"

   $AdminApp       Object to allow application object to be managed
   install         AdminApp command install
   <location EAR>  Name and the location of the EAR file to install
   {options}       Additonal options for the EAR file installation
```

The Example 8-1 shows the output of the installation.

Example 8-1 Command line output application install

```
Installing of the J2EE Enterprise Application ITSOBankEAR
WASX7209I: Connected to process "server1" on node dhcp39065 using SOAP
connector;  The type of process is: UnManagedProcess
ADMA5016I: Installation of ITSOBankEAR started.ADMA5005I: Application
ITSOBankEAR configured in WebSphere repository
ADMA5001I: Application binaries saved
ADMA5011I: Cleanup of temp dir for app ITSOBankEAR done.
ADMA5013I: Application ITSOBankEAR installed successfully.
```

The application is now successfully installed in the *installedApps/<nodename>*- directory. If the configuration of the Application Server has changed, you have to save the changes with the **$AdminConfig** object command:

```
# wsadmin.sh -c "\$AdminConfig save"

   $AdminConfig    Object manipulates configuration data for a WebSphere
                   installation
   save            AdminConfig command to save the configuration
```

Example 8-2 Command line output configuration save

```
Save configuration
WASX7209I: Connected to process "server1" on node dhcp39065 using SOAP
connector;  The type of process is: UnManagedProcess
```

8.3.2 Control the Application Server

The WebSphere Application Server contains several command line tools that can be used to start, stop, and monitor WebSphere Application Server processes and nodes. These tools work on local server and nodes. They cannot operate on a remote server or node. In the section before you used the wsadmin tools as command line tool to install the application. Now we want to show how to start and to stop the Application Server without the Administrator Console.

Start the Application Server

Make sure, you have the access to run the shell scripts under the bin directory of the WebSphere Application Server. To start the Server run the *startServer.sh* script:

```
# /opt/WebSphere/AppServer/bin/startServer.sh <servername>
```

The server is started when the following message appears:

```
Server <servername> open for e-business; process id is <PID>
```

Stop the Application Server

Similar to the start of the Application Server run the Shell script *stopServer.sh* to stop the server.

```
# /opt/WebSphere/AppServer/bin/stopServer.sh <servername>
```

The server is started when the following message appears:

```
Server <servername> stop completed.
```

8.3.3 Deployment with Ant

This section describes the combination of the command line tools wsadmin and the build tool Ant. This build file creates the ITSO Bank EAR file to deploy it on an Application Server. You know how to handle the wsadmin tool to install a new Enterprise Application and to *stop* and *start* the Application Server via the command line. This will be realized only with the *<exec>* task in Ant. See Figure 8-24. The build script is complete for the build and deployment process.

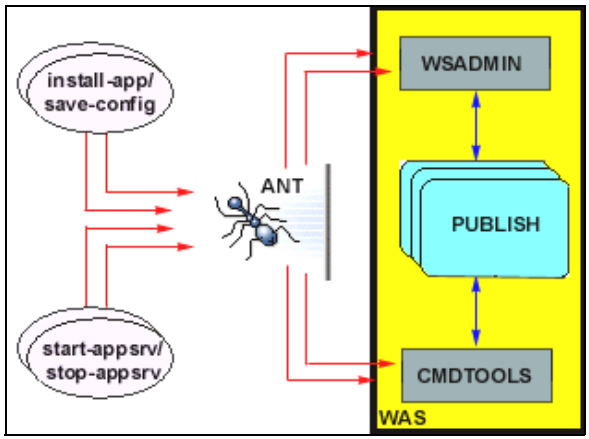


Figure 8-24 Ant invokes the Application Server tools

Update the properties file

The properties file is the important configuration file of the build process. It includes all needed properties that are used in the build process of the operating system. To set the properties for the *<exec>* task, you have to add to your build properties the following variables:

```
# Properties of the WebSphere Application Server
was.bin=  /opt/WebSphere/AppServer/bin
was.admin= wsadmin.sh
was.stop= stopServer.sh
was.start= startServer.sh
appsrv.name= server1
ear.install = \$AdminApp install ${dest.ear}
ear.uninstall = \$AdminApp uninstall ${ear}
save.config  = \$AdminConfig save
```

In the next steps we want to add and explain the additional targets to the build file *build.xml*.

First, you have to change in top of the project name the default target attribute from “ear” to “start-appsrv”. This target is our last target to install the application, and it depends on all the others. See below:

```
<project name="ITSOBankingExample" default="start-appsrv">
```

Note: The Application Server should be up before you are running the Ant script to install the Enterprise Application, otherwise, the service is not available to install.

Target “install-app”

The target “install-app” uses the <exec> task to start the installation with running the wsadmin tool on the command line. This target depends on the target “ear” to start the installation:

```
dir          Installation bin-directory of the Application Server
executable   Name of the Shell script wsadmin.sh
arg value    Using the command task to run the wsadmin tool (-c)
arg value    Object AdminApps, location and name of the EAR file to install
```

Example 8-3 Target “install-app”

```
<!-- Target install the J2EE Enterprise Application ITSOBankEAR -->
   <target name="install-app" depends="ear">
      <echo message="Installing of the J2EE Enterprise Application ${ear}" />
        <exec dir="${was.bin}" executable="${was.admin}" >
        <arg value="-c" />
        <arg value="${ear.install}" />
        </exec>
   </target>
```

Target “save-config”

The target “save-config” uses the *<exec>* task to save the configuration of the Application Server with running the wsadmin tool on the command line. You have to run this task by every changes of the Application Server. This target depends in our case on the target “install” to save the configuration changes:

```
dir          Installation bin-directory of the Application Server
executable   Name of the Shell script wsadmin.sh
arg value    Using the command task to run the wsadmin tool (-c)
arg value    Object AdminConfig to save the configuration changes
```

Example 8-4 Target “save-config”

```
<!-- Target save-config saves the configuration changes of the Application
Server -->
   <target name="save-config" depends="install-app">
      <echo message="Save configuration" />
      <exec dir="${was.bin}" executable="${was.admin}" >
         <arg value="-c" />
         <arg value="${save.config}" />
      </exec>
   </target>
```

Target “stop-appsrv”

The target “start-appsrv” uses the *<exec>* task to stop the Application Server on the command line with Ant, which uses the existing shell script on the *WebSphere/AppServer/bin* directory. It depends on the target “save-config”.

```
dir          Installation bin-directory of the Application Server
executable   Name of the Shell script stopServer.sh
arg value    Name of the Application Server to stop
```

Example 8-5 Target “stop-appsrv”

```
<!-- Target stop Application Server -->
   <target name="stop-appsrv" depends="save-config">
      <echo message="Stop Application Server" />
      <exec dir="${was.bin}" executable="${was.stop}" >
         <arg value="${appsrv.name}" />
      </exec>
   </target>
```

Target “start-appsrv”

The target “start-appsrv” uses the *<exec>* task to start the Application Server on the command line with Ant. This target depends on the target “stop” to start the Application Server again.

```
dir          Installation bin-directory of the Application Server
executable   Name of the Shell script startServer.sh
arg value    Name of the Application Server to start
```

Example 8-6 Target “start-appsrv”

```
<!-- Target start Application Server -->
   <target name="start-appsrv" depends="stop-appsrv">
      <echo message="Start Application Server" />
      <exec dir="${was.bin}" executable="${was.start}" >
         <arg value="${appsrv.name}" />
      </exec>
   </target>
```

Run Ant

The build file is now modified and can be run on the command line in the build directory. The output of the StdOutput and StdError can be stored in log files:

```
# ant 1>buildOut.log 2>buildErr.log
```

or use the Ant option **-logfile** *<logfilename>*

```
# ant -logfile build.log
```

without any logging, run Ant like:

```
# ant
```

A sample output of the log file can be found under BuildwithAnt/Build directory.

Note: The build and properties files are located in the BuildwithAnt directory in the sample code.

The Enterprise Application is now successfully installed and can be accessed with the URL address.

A

Installation instructions

In this appendix, we discuss the following topics:

- How to install Linux shows you how to install Linux.
- How to install WebSphere Studio Application Developer.
- How to install WebSphere Application Server.
- How to install IBM DB2.
- How to configure CVS.
- How to configure Telnet, FTP, and Samba and other useful tools.

© Copyright IBM Corp. 2003. All rights reserved.

How to install Linux

Follow these steps to install Linux:

1. Boot the computer using the Red Hat Linux 8.0 CD-ROM.
2. At the boot prompt press Enter.
3. In the Welcome window, click **Next.**
4. In the Language Selection window choose **English** and click **Next.**
5. In the Keyboard Configuration window choose **U.S. English** and click **Next.**
6. In the Mouse Configuration window choose **defaults** and click **Next.**
7. In the Install Type window choose **Custom** and click **Next.**
8. In the Disk Partition Step window choose **Automatically partition** and click **Next.**
9. In the Automatic Partitioning window choose to **Remove all partition on this system** and click **Next.**
10. Confirm `deletion of partitions` and click **Next.**
11. In the Disk Setup window choose **defaults** and click **Next.**
12. In the Boot Loader Configuration window choose **defaults** and click **Next.**
13. In the Network Configuration window, select the required Network devices and DHCP or Static addressing and click **Next.**
14. In the Firewall Configuration window choose **Trusted Devices**, select the **default network card,** allow all incoming protocols, and click on **Next.**
15. In the Additional Language Support window choose the defaults and click **Next.**
16. In the Time Zone Section window choose your time zone and click **Next.**
17. In the Account Configuration window enter your root password then click **Add** to add a user account and click **Next.**
18. In the Authentication Configuration window choose the defaults and click **Next.**
19. In the Package Group Selection window go to the Desktops section, under the X Windows System section de-select GNOME. Ensure that **KDE** is selected. In the Applications section choose the **defaults**. In the Development section under Development Tools, ensure that **CVS** is selected by default and select **expect**. In the System section choose the **Administration Tools** and in System Tools select **VNC.** Click the **Select Individual Packages** check box and then click **Next.**

20. In the Individual Package Selection window, the User Interface section under the X section select **VNC-server**. In the System Environment section under Shells section select **pdksh** and **zsh**. In the Daemons section select **wu-ftpd** and ensure **samba** has already been selected. In Development section under the Tools select **cervisia**. In the Languages section select **expect**. Click **Next** to continue.
21. In the Unresolved Dependencies window select **OK** and click **Next.**
22. In the About to Install window click **Next.**
23. Choose **to create a boot disk** and click **Next** to continue.
24. In the Graphical Interface Configuration window choose the defaults and click **Next.**
25. In the Monitor Configuration window choose the defaults and click **Next.**
26. In the Customized Graphics Configuration window choose the defaults and click **Next** to reboot.

You can run Red Hat Update Agent to update the packages form the Linux Web site.

How to install WebSphere Application Developer

Follow these steps to install WebSphere Application Developer 5.0:

1. As root, you can run Application Developer by typing the following command: **`./install.sh`**
2. Select your language.
3. Select **1** for the License Agreement screen.
4. Select following runtime environments, press the Enter or y key to continue:
 a. WebSphere Application Server, Version 4
 b. WebSphere Application Server, Version 5
 c. WebSphere Application Server Express, Version 5
5. Select to install the plug-in samples or not.
6. Select to install Rational ClearCaseLT or not.
7. Go though the product registration process.
8. Look for the `installed successfully` message.
9. Once completed, launch the application, and confirm connectivity.

Sample installation log:

```
[root@dhcp39035 IBMWSAppDev-5.0-0]# ./install.sh
```

```
---------------------------------------------------------------------
WebSphere Studio Application Developer for Linux 5.0.0
---------------------------------------------------------------------

Software Licensing Agreement
1. English
2. Danish
3. Dutch
4. Finnish
5. French
6. German
7. Italian
8. Norwegian
9. Polish
10. Portuguese
11. Russian
12. Spanish
13. Swedish

Please enter the number that corresponds to the language
you prefer.

1
Software Licensing Agreement
Press Enter to display the license agreement on your
screen. Please read the agreement carefully before
installing the Program. After reading the agreement, you
will be given the opportunity to accept it or decline it.
If you choose to decline the agreement, installation will
not be completed and you will not be able to use the
Program.


International Program License Agreement


Part 1 - General Terms


PLEASE READ THIS AGREEMENT CAREFULLY BEFORE USING THE
PROGRAM. IBM WILL LICENSE THE PROGRAM TO YOU ONLY IF YOU
FIRST ACCEPT THE TERMS OF THIS AGREEMENT. BY USING THE
PROGRAM YOU AGREE TO THESE TERMS. IF YOU DO NOT AGREE TO
THE TERMS OF THIS AGREEMENT, PROMPTLY RETURN THE UNUSED
PROGRAM TO THE PARTY (EITHER IBM OR ITS RESELLER) FROM WHOM
YOU ACQUIRED IT TO RECEIVE A REFUND OF THE AMOUNT YOU PAID.

The Program is owned by International Business Machines
Corporation or one of its subsidiaries (IBM) or an IBM
```

```
Press Enter to continue viewing the license agreement, or,
Enter "1" to accept the agreement, "2" to decline it or
"99" to go back to the previous screen.

1
Exited with: 9

-------------------------------------------------------------------------
Select the optional program features you want installed.

=> WebSphere Application Server v4.0 Run-Time Environment
(y/n) [ENTER for y]

=> WebSphere Application Server - Express v5.0 Run-Time Environment
(y/n) [ENTER for y]

=> Examples for Eclipse Plug-In Development
(y/n) [ENTER for y]

=> Rational ClearCase SCM Team Adapter
(y/n) [ENTER for y]
-------------------------------------------------------------------------

The following features will be installed:

WebSphere Studio Application Developer
WebSphere Application Server v4.0 Run-Time Environment
WebSphere Application Server v5.0 Run-Time Environment
WebSphere Application Server - Express v5.0 Run-Time Environment
Examples for Eclipse Plug-In Development
Rational ClearCase SCM Team Adapter

Press Enter to start the installation
-------------------------------------------------------------------------
Remove old product...
Installing WebSphere Studio Application Developer ...
- install rpm/IBMWSWB-5.0-0.i386.rpm
- install rpm/IBMWSTools-5.0-0.i386.rpm
- install rpm/IBMWSSiteDevExp-Core1-5.0-0.i386.rpm
- install rpm/IBMWSSiteDevExp-Core-5.0-0.i386.rpm
- install rpm/IBMWSSiteDev-Core-5.0-0.i386.rpm
- install rpm/IBMWSAppDev-Core-5.0-0.i386.rpm
- install rpm/IBMWSAppDev-Product-5.0-0.i386.rpm
- install rpm/IBMWSAppDev-5.0-0.i386.rpm
Installing WebSphere Application Server v4.0 Run-Time Environment ...
- install rpm/IBMWSTools-WAS-AES-V4-5.0-0.i386.rpm
Installing WebSphere Application Server v5.0 Run-Time Environment ...
- install rpm/IBMWSTools-WAS-BASE-V5-5.0-0.i386.rpm
```

```
Installing WebSphere Application Server - Express v5.0 Run-Time Environment ...
- install rpm/IBMWSTools-WASExpress-BASE-V5-5.0-0.i386.rpm
Installing Examples for Eclipse Plug-In Development ...
- install rpm/IBMWSWB-samples-5.0-0.i386.rpm
Installing Rational ClearCase SCM Team Adapter ...
- install rpm/RatlCCSCMAdapter-2.1-0.i386.rpm
Configure ...

------------------------------------------------------------------------
IBM Product Registration

Do you want to register now - Netscape or Mozilla are required.
(y/n) [ENTER for y] n

WebSphere Studio Application Developer has been installed successfully
```

How to install WebSphere Application Server

Follow these steps to install WebSphere Application Server 5.0:

1. As root, you can run Application Server by typing the following command **`./LaunchPad.sh`** from the install directory or use a silent install command, **`./install.sh <response_file>`**. We have used the first installation method.
2. In the Install Wizard window, click **Next.**
3. In the Licence Agreement window choose to read and accept the agreement and click **Next.**
4. Once checking is complete, click **Next.**
5. Choose custom install, click **Next.**
6. You have an option to remove or add components, we used the defaults.
7. Click **Next** to continue.
8. Use the default directory /opt/WebSphere/AppServer for WebSphere Application Server and /opt/IBMHttpServer for the IBM Http Server, click **Next.**
9. In the Node name window choose the defaults, click **Next.**
10. In the Installation window review the packages and click **Next** to continue.
11. Once completed, launch the application and confirm connectivity.

How to install IBM DB2

Follow these steps to install IBM DB2 Version 8.0:

Login into the Linux operating system using your account and run X Windows; to un-compress a tar file use the **`tar -xvf filename.tar`** command.

1. As root, run DB2 by typing the following command **`./db2setup`** from the install directory.
2. In the IBM DB2 Setup Launchpad window, click **Install Products**.
3. In the Setup window ensure DB2 UDB Enterprise Server Edition is selected and click **Next**.
4. Figure A-1 will appear.



Figure A-1 DB2 Setup started window

5. In the Welcome to DB2 Setup Wizard window, click **Next**.
6. In the Software Licence Agreement window read and accept the agreement and click **Next**.
7. In the Setup window choose **custom** and click **Next**.
8. In the Select the installation action window select the defaults and click **Next**.

9. In the Select the Features to install window, select **Client Support** section and check-off **xml extender**. In Administration Tools section, check-off **db2 Web** tools. In the Application Development tools section, check-off **development centre** and **ADT sample programs,** then click **Next.**
10. In the Languages window select the **defaults** and click **Next.**
11. In the Set user info for DB2 admin server window, enter the username, group name, and password for the admin user and click **Next.**
12. In the Setup up a DB2 instance window, select the **defaults,** and click **Next.**
13. in the Select how the installation will be used window, select the **defaults**, and click **Next.**
14. In the Instance owner window, enter the username, group name, and password for the instance user and click **Next.**
15. In the Fenced owner window, enter the username, group name, and password for the fenced user, and click **Next.**
16. In the Configuring DB2 TCP/IP communication window, select the defaults, and click **Next.**
17. In the Set instance properties window, select the defaults, and click **Next.**
18. In the Prepare the DB2 tools catalog window, select to use the local database, and click **Next.**
19. In the Specify a local database window, select the **defaults**, and click **Next.**
20. In the Setup admin contact window, update the contact information, or select the **defaults**, and click **Next.**
21. In the Start to copy files window, click **Finish.**

Figure A-2 will appear.

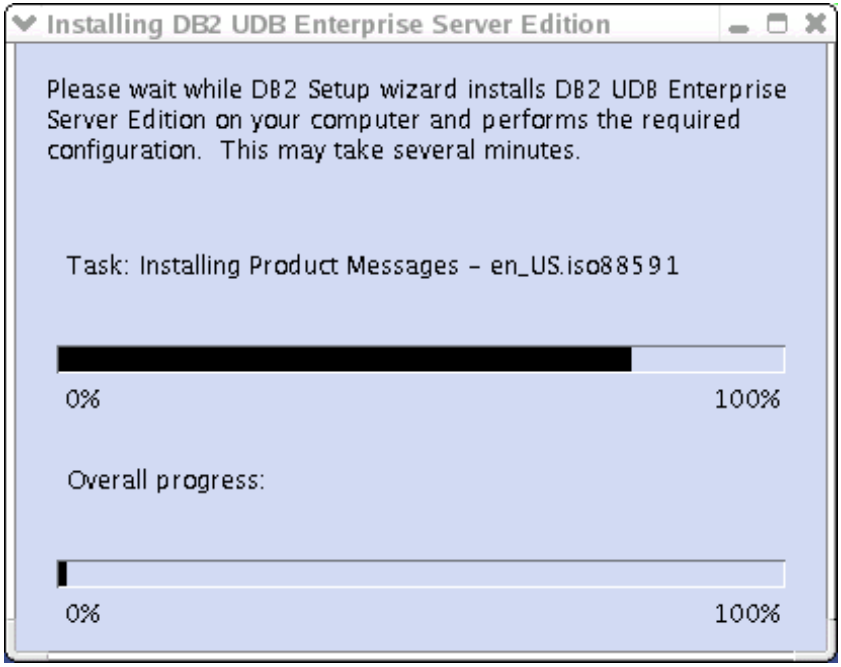


Figure A-2 DB2 Installing window

Once the setup is completed, Figure A-3 will appear.

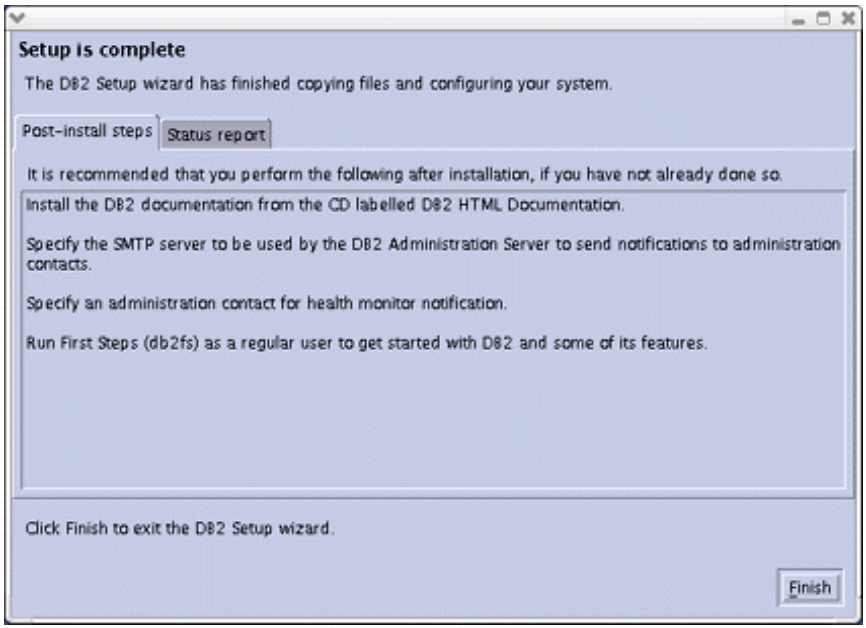


Figure A-3 DB2 Setup finished window

22. Login in as the instance user and run First Steps. Create the sample database to confirm the DB2 installation.

How to configure CVS

Follow these steps to configure a version control system:

1. As root, create accounts for CVS and other users.
2. Add the users to the CVS group.
3. As CVS, in the home directory make cvs_rep.
4. Add the following variables to the bash.profile file:

```
CVS=/home/cvs/crs_rep
export CVSROOT
```

5. Issue source **`.bash_profile`** command to use the updated profile.
6. Issue **`cvs init`** command to create a new repository.
7. If your system uses inetd then add the following variable to the */etc* file:

```
cvspserver 2401/tcp
```

8. Also, change `2401` to `cvspserver` in the inetd.conf file.

9. If your system uses *xinetd* then create a file called *cvspserver* under the */etc* directory. The contents of the file are:

```
services cvspserver {
port = 2401
socket_tupe = stream
protocol = tcp
wait = no
user = root
passenv = PATH
server = /user/local/bin/cvs
server_args = -f --allow-root=/user/cvsroot pserver
}
```

10. Re-start to start these services.

How to configure Telnet, FTP, and Samba

Follow these steps to configure Telnet:

1. Open a terminal window.
2. As root, edit telnet file under /etc/xinetd.d. Change the disable parameter to no.

Follow these steps to configure FTP:

1. Open a terminal window.
2. As root, edit wu-ftpd file under /etc/xinetd.d. Change the disable parameter to no.

Follow these steps to configure Samba:

1. Open a terminal window.
2. Go to the /etc/samba/, at the prompt enter **`touch smbpasswd`**
3. Change the workgroup parameter to your workgroup in the smb.conf file.
4. Open up the tmp directory by un-commenting these variables in the smb.conf file.
5. Go to the system setting in the services configuration, activate smb agent and reboot the machine.

B

Additional material

This redbook refers to additional material that can be downloaded from the Internet as described below.

Locating the Web material

The Web material associated with this redbook is available in softcopy on the Internet from the IBM Redbooks Web server. Point your Web browser to:

```
ftp://www.redbooks.ibm.com/redbooks/SG246431/sg246431.tar
```

Alternatively, you can go to the IBM Redbooks Web site at:

```
ibm.com/redbooks
```

Select **Additional materials** and open the directory that corresponds with the redbook form number SG246431.

Using the Web material

The additional Web material that accompanies this redbook includes the following file:

File name	*Description*
SG246431.tar	Sample codes and database scripts

© Copyright IBM Corp. 2003. All rights reserved.

System requirements for downloading the Web material

The following system configuration is recommended:

Hard disk space: 10 MB minimum
Operating System: Linux
Processor: Pentium III 600Mhz or above
Memory: 512MB

How to use the Web material

Create a subdirectory (folder) on your workstation, and unzip the contents of the Web material zip file into this folder.

Directories:

- BuildwithAnt: samples for Ant
- DBScripts: database generation scripts
- Deploy: Samples for deploy
- ServletJSP: Sample Web application
- XML: Sample XML application

Abbreviations and acronyms

AAT	application assembly tool
ACL	access control list
API	application programming interface
BLOB	binary large object
BMP	bean-managed persistence
CCF	Common Connector Framework
CICS	Customer Information Control System
CMP	container-managed persistence
CORBA	Component Object Request Broker Architecture
DBMS	database management system
DCOM	Distributed Component Object Model
DDL	data definition language
DLL	dynamic link library
DML	data manipulation language
DOM	document object model
DTD	document type description
EAB	Enterprise Access Builder
EAI	Enterprise Application Registration
EAR	enterprise archive
EIS	Enterprise Information System
EJB	Enterprise JavaBeans
EJS	Enterprise Java Server
FTP	File Transfer Protocol
GUI	graphical user interface
HTML	Hypertext Markup Language
HTTP	Hypertext Transfer Protocol
IBM	International Business Machines Corporation
IDE	integrated development environment
IDL	Interface Definition Language
IIOP	Internet Inter-ORB Protocol
IMS	Information Management System
ITSO	International Technical Support Organization
J2EE	Java 2 Enterprise Edition
J2SE	Java 2 Standard Edition
JAF	Java Activation Framework
JAR	Java archive
JDBC	Java Database Connectivity
JDK	Java Developer's Kit
JFC	Java Foundation Classes
JMS	Java Messaging Service
JNDI	Java Naming and Directory Interface
JSDK	Java Servlet Development Kit
JSP	JavaServer Page
JTA	Java Transaction API
JTS	Java Transaction Service
JVM	Java Virtual Machine
LDAP	Lightweight Directory Access Protocol
MFS	message format services
MVC	model-view-controller
OLT	object level trace
OMG	Object Management Group
OO	object oriented

© Copyright IBM Corp. 2003

OTS	object transaction service
RAD	rapid application development
RDBMS	relational database management system
RMI	Remote Method Invocation
SAX	Simple API for XML
SCCI	source control control interface
SCM	software configuration management
SCMS	source code management systems
SDK	Software Development Kit
SMR	Service Mapping Registry
SOAP	Simple Object Access Protocol (a.k.a. Service Oriented Architecture Protocol)
SPB	Stored Procedure Builder
SQL	structured query language
SRP	Service Registry Proxy
SSL	secure socket layer
TCP/IP	Transmission Control Protocol/Internet Protocol
UCM	Unified Change Management
UDB	Universal Database
UDDI	Universal Description, Discovery, and Integration
UML	Unified Modeling Language
UOW	unit of work
URL	uniform resource locator
VCE	visual composition editor
VXML	voice extensible markup language
WAR	Web application archive
WAS	WebSphere Application Server
WML	Wireless Markup Language
WS	Web Service
WSBCC	WebSphere Business Components Composer
WSDL	Web Service Description Language
WSTK	Web Service Development Kit
WTE	WebSphere Test Environment
WWW	World Wide Web
XMI	XML metadata interchange
XML	eXtensible Markup Language
XSD	XML schema definition

Related publications

The publications listed in this section are considered particularly suitable for a more detailed discussion of the topics covered in this redbook.

IBM Redbooks

For information on ordering these publications, see “How to get IBM Redbooks” on page 194.

- *Linux for WebSphere and DB2 Servers,* SG24-5850
- *Linux Web Hosting with WebSphere, DB2 and Domino,* SG24-6007
- *WebSphere Application Server V4 for Linux - Implementation and Deployment Guide*
- *IBM Framework for e-business - Technology, Solution and Design Overview,* SG24-6248
- *Web Services Wizardry with WebSphere Studio Application Developer,* SG24-6292
- *WebSphere Version 4 Application Development Handbook*, SG24-6134
- *IBM WebSphere V4.0 Advanced Edition Handbook*, SG24-6176
- *EJB Development with VisualAge for Java for WebSphere Application Server,* SG24-6144
- *WebSphere Application Server Enterprise Edition 4.0 - A Programmer’s Guide,* SG24-6504

Other resources

These publications are also relevant as further information sources:

- *Learning Red Hat Linux, 2nd Edition, O’Reilly & Associates*
- Core servlets and JavaServer Pages, Sun Microsystems & Associates

© Copyright IBM Corp. 2003. All rights reserved.

Referenced Web sites

These Web sites are also relevant as further information sources:

- GNOME - Computing made easy

 http://www.gnome.org

- K Desktop Environment

 http://www.kde.org

- IBM and Linux

 http://www.ibm.com/linuxDescription1

- IBM Framework for e-business

 http://www.ibm.com/software/ebusiness

- IBM WebSphere Software platform

 http://www.ibm.com/software/websphere

- Java servlet technology

 http://java.sun.com/products/servlet

- WebSphere Developer Domain

 http://www7b.software.ibm.com/wsdd

How to get IBM Redbooks

You can order hardcopy Redbooks, as well as view, download, or search for Redbooks at the following Web site:

ibm.com/redbooks

You can also download additional materials (code samples or diskette/CD-ROM images) from that site.

IBM Redbooks collections

Redbooks are also available on CD-ROMs. Click the CD-ROMs button on the Redbooks Web site for information about all the CD-ROMs offered, as well as updates and formats.

Index

A
Account table 54
Address table 54
Admin client 62
Administration Console 153, 159
Ant 4
Apache Ant 137
Application Client Project 29
Application Developer 22, 41
Application Server 172

B
BankAccount 34
Bean Managed Persistence Entity Bean 78
BMP 78, 97
Body 42
Business Integration 10

C
Cascading stylesheet 13
CheckingAccount 34
CICS 8
ClearCase 137
ClearCase LT 13
Component Object Request Broker 6
Connection factory JNDI name 64
Console 18
Controller Page 40
Create a new Front Controller 40
Create A New SQL Statement 56
Customer Account table 54
Customer table 53
CVS 27, 137

D
Data perspective 24
Database Access Objects 86
Database connection 44
DB Explorer 23
DB Servers 56, 118
DB2 36
DB2 connection 49
db2_jdbc_driver_path 159
DBConnectionSpec 45
DDL 24
Debug perspective 24
Default bindings 160
Deploy and RMIC Code 93
Deployment with Ant 172
destination folder 40
Destination JNDI name 64
dispatch 48
display-name tag 49
doGet 47
Domino Designer 9
doPost 47, 124, 126
DTD 13
DTD Editor 122

E
EAR 22, 38, 94, 152
EAR Project 29
Eclipse Framework 9
EJB 29, 34, 69
EJB container 61
EJB Project 29
ejbCreate 90
ejbFindByPrimaryKey 91
ejbLoad 92
ejbStore 92
EJB-to-RDB mapping tools 12
Embedded JMS Provider 61
Enable administration client 62
Enterprise Application Project 29
Enterprise Java services 8
Enterprise JavaBeans 4, 8
Enterprise Modernization 11
Entity Beans 60
execute 51
Export 152
Expression Builder 57

F
Firewall 169
Framework 6–7

© Copyright IBM Corp. 2003. All rights reserved.

G
Gallery 20
garbage collection 25
Generate new XML 119
GNOME 3

H
Header 41
Hierarchy 19
HTML 20, 28, 36, 38, 41, 47, 115
Html 40
HTTP 43
HTTP port number 168
HttpServletRequest 47
HttpServletResponse 47
HttpSession 44
Hypertext Transfer Protocol 8

I
IBM Agent Controller 22, 163
IBM Database Access Tag Library 40
IBM DB2 App Driver 56
IBM extension deployment descriptor 71
init 126
initializer 51
Integrated Development Environment 16
Internal JMS Server 64
Internet Inter-ORB Protocol 6
Internet Message Access Protocol 6
Invocation Internet Inter-ORB Protocol 13
ITSO Bank database 53
ITSO bank model 33
ITSO banking example xv, 4
ITSOBankEJB 33
ITSOBankSelectEAR 38
ITSOBankSelectWeb 36–38, 40, 43
ITSOBankUpdateWeb 40
ITSOBankUpdatWeb 36
ITSOBankWeb 41, 43, 46, 50

J
J2C Authentication Data Entries 158
J2EE 37, 69, 136, 140
J2EE 1.3 12
J2EE deployment descriptors 21
J2EE perspective 21
JAR 29, 38
Java Bean Web Pages wizard 50
Java Management Extensions 14
Java Message Driven Beans 60
Java Messaging Services 13
Java perspective 18
Java project 29
JavaBean 51
JavaBeans 4, 36
JavaServer Pages 8
JDBC 2.0 89
JDBC Provider 155
JMS 72
JMS Client 74
JMS consumer 61
JMS Servers 64
JNDI 94
JSP 4, 20, 28–29, 34, 36, 38, 40, 43, 52
JSP SQL tags 44

K
KDE 3

L
listener port 65
Listener Ports 63

M
Message Beans 13
Message Driven Beans 4
Message Listener Service 63
Message-Driven Bean 68
Message-Driven Beans 60
Meta 41
MQJD Provider 62

N
Navigator 21–22, 28
Navigator views 18
nextPage 48

O
onMessage 73
onMessage() method 73
OrderCheck 34
Out Of Memory exception 162
Outline 21, 28

P

Packages 19
PASV Mode 169
performServices 48
performTask 47–48
Pervasive Products 11
Point-To-Point Messaging 60
Post Office Protocol 6
primary key 90
Profiling perspective 25
profiling tools 25
programming model 8
Properties 21, 28
Publish/Subscriber Messaging 60

R

RDB to XML mapping 116–117, 119
Red Hat 3
Redbooks Web site 194
 Contact us xvii
Remote Server 162
Re-order Check table 54
Resource perspective 17
RMI-IIOP 94
Run administrative client 63
Run on Server 40
Runtime Connection Page 40

S

SavingsAccount 34
SCM 13
Secure Network 9
Secure Socket Layer 6
Security Programming Interfaces 14
SELECT 56
Select Statement from the SQL Statement Type 40
Server Components 64
Server Configuration 22
Server perspective 22, 62
Server Project 29
Servlet 4
servlet tag 49
servlet-mapping tag 50
servlets 8, 36
Session 44
Session Beans 60
Simple Mail Transfer Protocol 6
Simple Object Access Protocol 13
SQL 53, 58
SQL builder 117
SQL command 51
SQL query 39
SQL Query builder 56
SQL statement 39, 45, 115
SQL tag library 43
SQL to XML wizard 115
SQLtoXML 32, 115
Start the Server 154
Struts 132
SuSE 3

T

Tables 57
taglib 43
taglib tag 50
targets 141
Tasks 18, 21, 28
Team perspective 26
Thin client 7
Three-tier computing model 7
Transaction Servers 10
Transfer balance 46
Transmission Control Protocol or Internet Protocol 6
TransRecord 34
TSX 45
Turbolinux 3

U

UDDI Version 2 12
updateBean 50

W

WAR 29
WASQueue entries 67
Web application 21
Web application server 7
Web Deployment Descriptor 46, 49
Web perspective 19
Web Project 29
Web Project Features 37
Web Services Description Language 13
web-app tag 49
WEB-INF 56
WebSphere Commerce 11

WebSphere Host Integration 11
WebSphere JMS Provider 64
WebSphere JMS Provider Options 65
WebSphere Portal 11
WebSphere pyramid 10
WebSphere Queue Destinations 65
WebSphere Studio 11
Workbench 16
wsadmin tool 170

X

Xalan processor 129
XHTML 13
XML 4, 13
XML Editor 121
XML form SQL query 116
XML from An SQL Query 119
XML from an SQL query 115
XML perspective 23
XMLtoSQL 115
XPath 13
XSL debugger 129
XSL debugging 115
XSL Editor 122
XSLT Processor 124
XSLT processor 32

(0.2”spine)
0.17”<->0.473”
90<->249 pages



Linux Application Development Using WebSphere Studio



A comprehensive guide to Linux support of WebSphere development tools

Develop, test, and deploy your Web application on Linux

Setting up your Linux enviroment

Linux is the fastest-growing server operating system in the world because of its powerful functionality, rock-solid stability, and open source foundation. Applications developed on Linux are reliable, portable, and cost efficient. This IBM Redbook helps you get familiar with IBM middleware and tools for Linux, and develop your new Web application on Linux.

This redbook is aimed to show IBM's ability to provide an advanced platform for WebSphere application development using Linux as the operating system.

The approach we have taken is to build an ITSO Banking example that has a backend database, and a frontend e-business banking application. The Linux distribution that we use is Red Hat Linux Version 7.3.

This book also shows you how to install the software to set up your development environment.

INTERNATIONAL TECHNICAL SUPPORT ORGANIZATION

BUILDING TECHNICAL INFORMATION BASED ON PRACTICAL EXPERIENCE

IBM Redbooks are developed by the IBM International Technical Support Organization. Experts from IBM, Customers and Partners from around the world create timely technical information based on realistic scenarios. Specific recommendations are provided to help you implement IT solutions more effectively in your environment.

For more information:
ibm.com/redbooks

SG24-6431-00 ISBN 0738427918